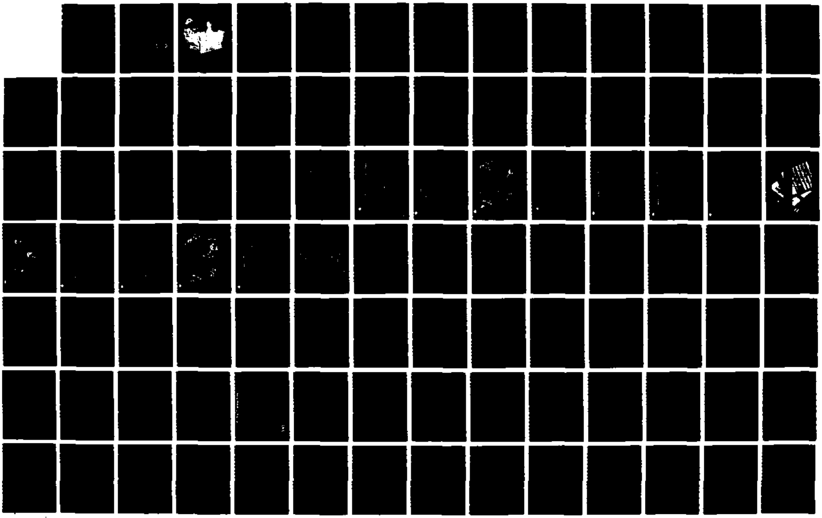
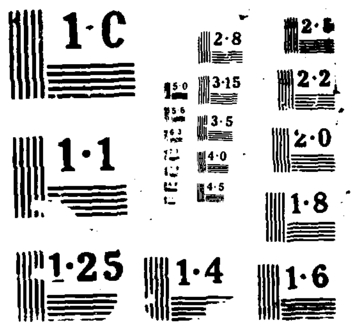
AD-A189 658 ANNUAL ASEET (ADA SOFTWARE ENGINEERING EDUCATION AND TRAINING) SYMPOSIUM (SLIDES) HELD IN DALLAS TX ON 9-11 JUNE 1987(U) ADA JOINT PROGRAM OFFICE ARLINGTON VA 1/4

UNCLASSIFIED 11 JUN 87 F/G 12/5 NL




1·1
1·25
1·4
1·6
1·8
2·0
2·2
2·8
3·15
3·5
4·0
4·5

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

DTIC FILE COPY

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Second Annual ASEET Symposium (SLIDES)		5. TYPE OF REPORT & PERIOD COVERED Tutorial, June 10-11, 1987
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) The Ada Software Engineering Education and Training (ASEET) Team		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION AND ADDRESS Ada Software Education and Training Team Ada Joint Program Office, 3E114, The Pentagon, Washington, D.C.20301-3081		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Ada Joint Program Office 3E 114, The Pentagon Washington, DC 20301-3081		12. REPORT DATE 9-11 June, 1987
		13. NUMBER OF PAGES 319
14. MONITORING AGENCY NAME & ADDRESS (If different from Controlling Office) Ada Joint Program Office		15. SECURITY CLASS (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

AD-A189 658

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20. If different from Report)

UNCLASSIFIED

DTIC ELECTE JAN 0 6 1988 S D

D

18. SUPPLEMENTARY NOTES

19. KEYWORDS (Continue on reverse side if necessary and identify by block number)

Ada Programming language, Ada Training, Education, Training, Computer Programs, Ada Joint Program Office, AJPO.

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This document contains prints of slides presented at the Ada Tutorial, Track I, Industry, and Track II, Academia, Keywords:

DD FORM 1473 1 JAN 73 EDITION OF 1 NOV 65 IS OBSOLETE

S/N 0102-LF-014-6601

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

SECOND ANNUAL ASEET SYMPOSIUM

9 - 11 JUNE 1987
DALLAS, TEXAS

PACKAGE SPECIFICATION

```
package Explore_Specs is
     type Wall_Enum is (n, s, e, w, u, d);
     type Exit_Type is (None, Open, Closed, Locked);
     type Item_Enum is (Weapons, Treasures, Keys, Misc);
     type Backpack_Item_Enum_Type is array (1..10)
           of Item_Enum;
     type Action_Type is (Take, Drop, Throw, None);
     subtype Name_Subtype is String (1..10);
     subtype Desc_Subtype is String (1..160);
     type Item_Action is array (1..5) of Action_Type;
     type Monster_Type is (Norm, Gary, Parker);
     type Backpack_Type is array (1..10) of Name_Subtype;
     type Danger_Type is array (1..2) of Name_Subtype;
     type Person is
     record
          Backpack : Backpack_Type;
          Backpack_Item_Enum : Backpack_Item_Enum_Type;
          Item_Count : Integer := 0;
     end record;

     type Wall_Type is
     record
          Wall_Exit : Exit_Type;
          Next_Room : Integer;
     end record;

     type Item_Desc is
     record
          Name   : Name_Subtype;
          Action : Item_Action;
          Lookup : Integer;
     end record;

     type Item_Type is array (1..3) of Item_Desc;
     type Room_Wall_Type is array (Wall_Enum) of Wall_Type;
     type Room_Item_Type is array (Item_Enum) of Item_Type;

     type Room_Type is
     record
          Wall : Room_Wall_Type;
          Item : Room_Item_Type;
          Description : Desc_Subtype;
          Dangers : Danger_Type;
     end record;

     Player : Person;
     Room   : array (1..25) of Room_Type;
```

PACKAGE SPECIFICATION (Continued)

```
--*************subprogram declarations*******

    function Length (Sentence : in String)
             return Integer;
    procedure Explore_Intro;
    procedure Open_Door (Room_Index :Integer);
    procedure Unlock_Door (Room_Index : Integer);
    procedure Check_Move (Room_Index : in Integer;
               Direction : in Wall_Enum;
               Ok        : out Boolean);
    procedure Describe (Room_Index : Integer);
    procedure Explore_Init;
    procedure Take_Item (Room_Index : Integer);
    procedure Drop_Item (Room_Index : Integer);

end Explore_Specs;
```

PACKAGE BODY

```
with Text_IO; use Text_IO;
package body Explore_Specs is

    function Length (Sentence : in String)
             return Integer is separate;
    procedure Explore_Intro is separate;
    procedure Open_Door (Room_Index :Integer)
              is separate;
    procedure Unlock_Door (Room_Index : Integer)
              is separate;
    procedure Check_Move (Room_Index : in Integer
                   Direction : in Wall_Enum;
                   OK        : out Boolean)
              is separate;
    procedure Describe (Room_Index : Integer)
              is separate;
    procedure Explore_Init is separate;
    procedure Take_Item (Room_Index : Integer)
              is separate;
    procedure Drop_Item (Room_Index : Integer)
              is separate;

end Explore_Specs;
```

MAIN ROUTINE

```
with Text_IO; use Text_IO;
with Explore_Specs; use Explore_Specs;
procedure Explore_Driver is

    -- type for commands


begin -- Explore_Driver

    -- the game


end Explore_Driver;
```

COMMAND_PACKAGE SPECIFICATION

```
package Command_Package is

   type Command_Type is private;

   procedure Get     (Command : out Command_Type);
   procedure Execute (Command : in Command_Type);
   function Done return Boolean;

   Bad_Command : exception;

private

   type Command_Type is access String;

end Command_Package;
```

MAIN ROUTINE

```
with Command_Package; use Command_Package;
procedure Play_the_Game is

   Command : Command_Type;

begin -- Play_the_Game

   COMMAND_LOOP:
   loop
       Get (Command);
   exit when Done;
       begin
         Execute (Command);
       exception
         when Bad_Command =>
            Put_Line "("Invalid command");
            Put_Line "("Enter another");
       end;
    end loop COMMAND_LOOP;

end Play_the_Game;
```

COMMAND_PACKAGE BODY

```
with Text_IO; use Text_IO;
with Game_Package; use Game_Package;
package body Command_Package is

   -- Execute parses the command
   procedure Parse (Command : in Command_Type;
                    Verb : out Verb_SubType;
                    Noun : out Noun_SubType)
             is separate;

   procedure Get (Command : out Command_Type)
             is separate;
   procedure Execute (Command : in Command_Type)
             is separate;
  function Done return Boolean is separate;

end Command_Package;
```

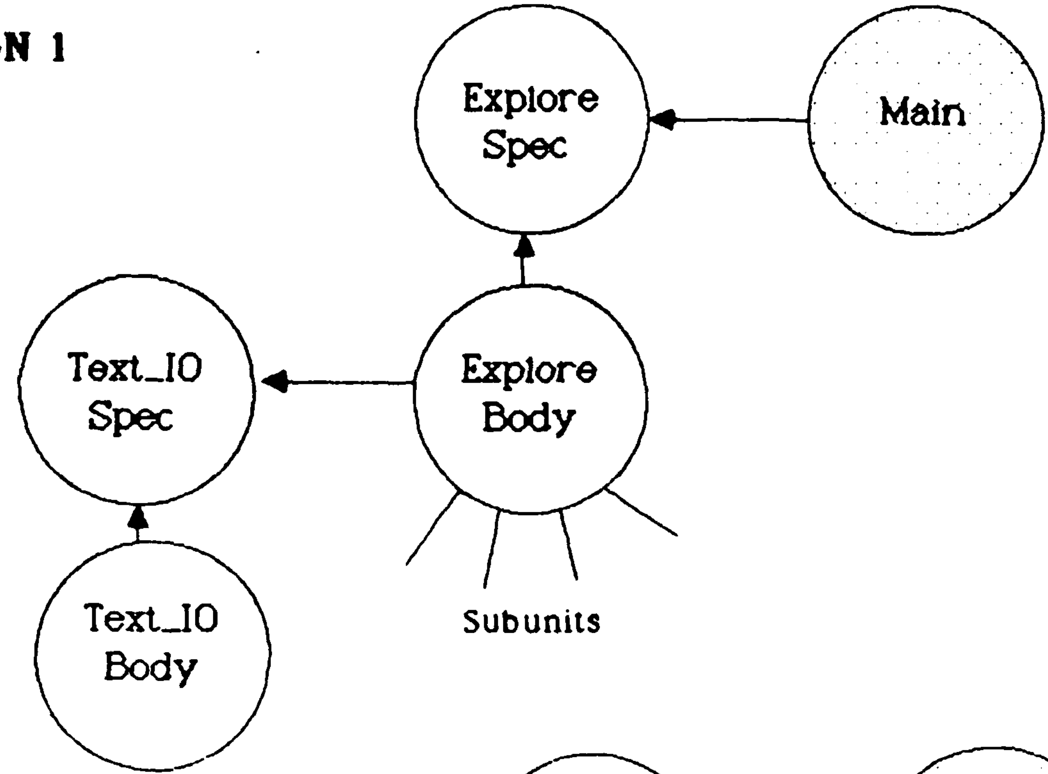
GAME_PACKAGE SPECIFICATION

```
package Game_Package is

   type Room_Names_Type is
        (Dungeon, Banquet_Hall, Kitchen,
         Throne_Room, Bedroom, Bathroom, None);
   type Words_Type is
        (move, pick_up, drop, display, stop,
         gold, diamonds, silver, Ada, game,
         room, my_status, north, east, south,
         west, up, down);
   subtype Verb_SubType is Words_Type
         range move .. stop;
   subtype Noun_SubType is Words_Type
         range gold .. down;
   subtype Treasures_SubType is Noun_SubType
         range gold .. Ada;
   subtype Directions_SubType is Noun_SubType
         range north .. down;

   procedure Move    (Where  : in Directions_SubType);
   procedure Pick_Up (Object : in Treasures_SubType);
   procedure Drop    (Object : in Treasures_SubType);
   procedure Display (Room   : in Room_Names_Type);
   procedure Display_Players_Status;

   No_Exit   : exception; --raised by Move
   No_Object : exception; --raised by Pick_Up or Drop

end Game_Package;
```

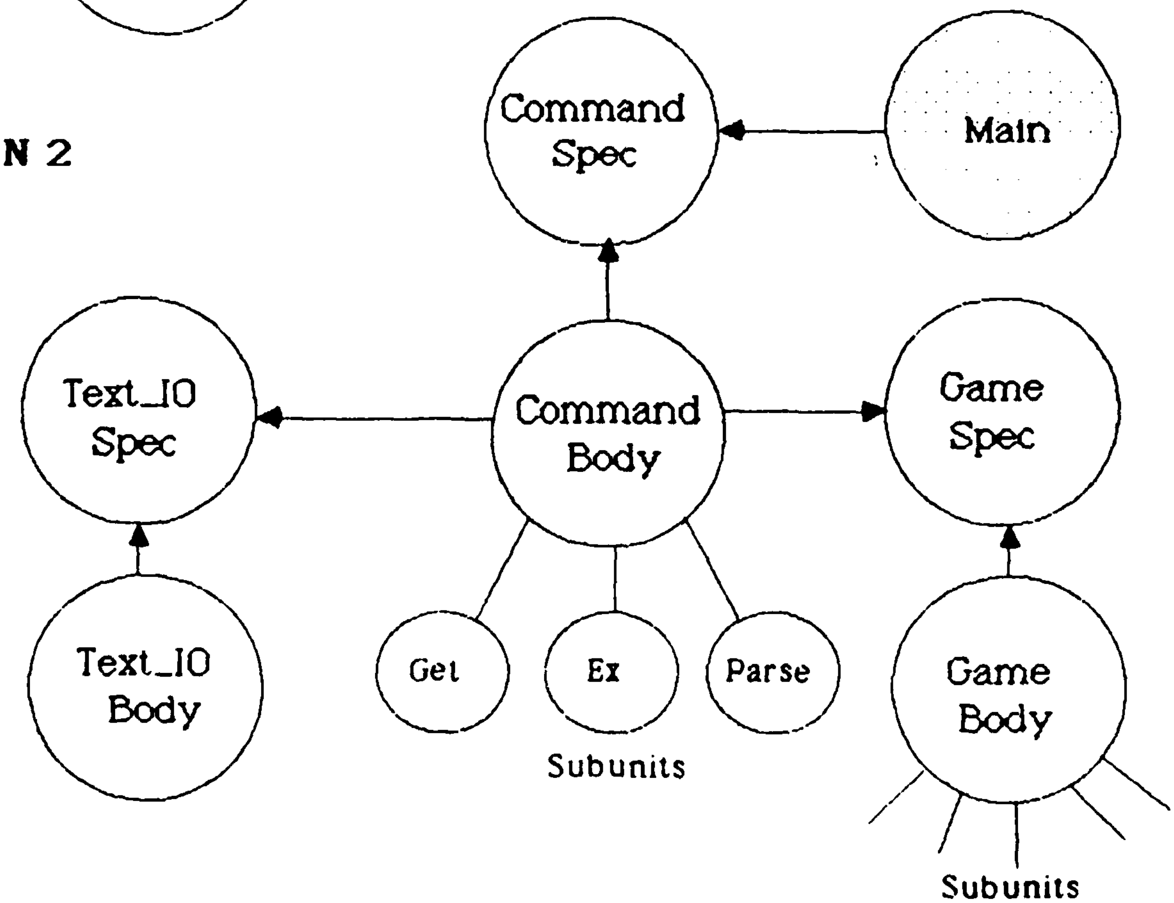
GAME_PACKAGE BODY

```
with Text_IO; use Text_IO;
package body Game_Package is

   type Exits_Type is array
        (Directions_SubType) of Room_Names_Type;

   type Treasures_Info_Type is
   record
      First_Time : Boolean := False;
      Points     : Positive;
   end record;
   Game_Treasures : Treasures_Info_Type;
   type Treasures_Set_Type is array
        (Treasures_SubType) of Treasures_Set_Type;

   Max_Points_per_Room : constant Postive := 50;
   type Rooms_Type is
   record
       Exits     : Exits_Type;
       Treasures : Treasures_Type;
       Points    : Positive range 1..Max_Points_per_Room;
       Message   : String (1..40);
   end record;

   type The_Game_Type is array
        (Room_Names_Type) of Rooms_Type;
   The_Game : The_Game_Type;

   Max_Points_per_Game : constant Positive := 500;

   type Player_Type is
   record
       Points    : Positive range 1..Max_Points_per_Game;
       Location  : Room_Names_Type;
       Treasures : Treasures_Set_Type;
   end record;
   Player : Player_Type;

   procedure Initialize_Game is separate;
   procedure Display_Initial_Greeting is separate;

   procedure Move    (Where : in Directions_SubType)
             is separate;
   procedure Pick_Up (Object : in Treasures_SubType)
             is separate;
   procedure Drop    (Object : in Treasures_SubType)
             is separate;
   procedure Display (Room : in Room_Names_Type)
             is separate;
   procedure Display_Players_Status is separate;
begin -- Game_Package
   Initialize_Game;
   Display_Initial_Greeting;
end Game_Package;
```

DESIGN 1



DESIGN 2



CONCLUSION

- CAN YOU FIND DESIGN FLAWS IN THE SECOND DESIGN??
- TECHNICAL MANAGERS CAN - AT THE END OF TWO DAYS!!

SLIDES

TRACK II - ACADEMIA

WEDNESDAY, JUNE 10, 1987

Treatment of the Ada Language in a
Programming Language (ACM-CS8) Course

D. M. Meeker

COURSE OBJECTIVES:

"(a) to develop an understanding of the organization of programming languages, especially the run-time behavior of programs;

(b) to introduce the formal study of programming language specification and analysis;

(c) to continue the development of problem solution and programming skills introduced in the elementary level material."

EXPLOITATION OF ADA'S STRONG TYPING.

Given:

- Most students are familiar with strong typing.
- Languages like Pascal allow only limited programmer exploitation of typing.
- Students need learn to appreciate and exploit derived data types.

Approach:

Learn to recognize and prevent typing errors using derived data types.

ADA ABSTRACTIONS

Typical Freshman and Sophomore coursework:

- Relatively simply software systems.

- Individually programmed assignments.

- Unfamiliar issues:

  information hiding.
  need to know.
  localization.
  maintainability.

- Data abstraction is primarily motivated by concerns to increase the understandability of code.

- Algorithmic abstraction introduced with structured programming and top down design concepts.

Lesson:

Data and algorithmic abstraction go hand in hand.

Approach:

Design and implement a data hierarchy.

FAULT TOLERANT ADA SYSTEMS

Typical student experiences:

- Never handled a data exception.
- Problem domains are small and strictly bounded.
- Algorithmic failure was a result of incorrect programming, and must be corrected by rewriting mainline code.
- All domain testing and error condition handling was incorporated directly into mainline code.

Ada Topics:

- Standard exceptions are always present.
- Data types may be constrained.
- The programmer may pretest domains with the "in" and "not in" operators.
- The programmer may control the exception handling with Ada code.
- The exception may be propagated to a higher level by deferring exception handling or re-raising the exception.
- Provision is made for premature exit from iterative processes.
- The programmer may create user defined exceptions.

TASKING AND CONCURRENCY IN ADA

Currently:

- Technological development is leading toward parallel processing.

- Almost all algorithmic approaches taught in the undergraduate curriculum are sequential.

TEACHING PARALLEL PROGRAMMING TECHNIQUES SHOULD BE A HIGH PRIORITY IN THE UNDERGRADUATE CURRICULUM.

Approach:

- Introduce tasks in Ada.

- The fact that the parallel software is run on sequential hardware system is irrelevant.

- Example programming assignment:

  calculate the maximum value of an array using a parallel divide and conquer technique.

SLIDES

Wednesday, 10 June 1987

Thursday, 11 June 1987

SLIDES

TRACK I - INDUSTRY

WEDNESDAY, JUNE 10, 1987

Managing the Implementation of an Ada® Training Program

June 10, 1987

Second Annual
Ada Software Engineering Education
and Training (ASEET) Team Symposium

Presented by:

Paul Barkowitz
Harris Corporation
Computer Systems Division
2101 West Cypress Creek Road
Fort Lauderdale, Florida 33309
(800) 245-6453

Harris Education Center

Overview of Harris

- Harris Corporation
  - Information Processing, Communications, and Electronics
  - $2.2 Billion in 1986 Sales
- Computer Systems Division
  - Engineering, Scientific, Educational, & Aerospace Industries
- Harris Education Center
  - Software and Hardware
  - Customers and Internal Students

Overview of Ada Training Issues

- Finding Ada Expertise
- Setting Goals and Objectives
- Determining Number & Length of Courses
- Producing Necessary Training Materials
- Judging Quality of Training

Finding Ada Expertise

- Existing Inhouse Expertise
  - Harris Ada Product Management Team
  - Software Development Staff
- Hire from Outside or Develop Expertise from Within
  - Scarcity of Ada Experts
  - Ada Syntax Expert /= Ada Software Engineer
  - Ada Expert /= Ada *Educational* Expert
- Decision to Develop Expertise from Within
  - Use Existing Educational Experts
  - Rely on Inhouse Experts for Assistance

Setting Goals and Objectives

- Initial Steering Force -- Joe Dangerfield
- Obtain Feedback From Users of Ada
- Develop Peer Review Relationship with Academic Institutions

Determining Number and Length of Courses

- More Courses = More Information = More $
- How to Integrate Software Engineering Principles
- Four-Course Curriculum
  - Introduction to Ada -- 1 Week
  - Advanced Ada -- 1 Week
  - Harris Ada Programming Support Environment (HAPSE®) -- 3 Days
  - Advanced Ada Programming Workshop -- 1 Week

Producing Necessary Training Materials

- Student Guide
- Laboratory Exercises
- Technical Documentation
- *Software Engineering with Ada* by Grady Booch
- Instructor Guide

Judging the Quality of Training

- Evaluation of Student's Performance
- Student Feedback
  - Wide Variety of Customers
  - Feedback from Harris Employees
- Academic Peer Review

The Ada® Training Guide

Priscilla J. Fowler

June 10, 1987

Software Engineering Institute

Carnegie-Mellon University
Pittsburgh, PA 15213

Sponsored by the U.S. Department of Defense

060887TR1A

Ada Transition Work To Date

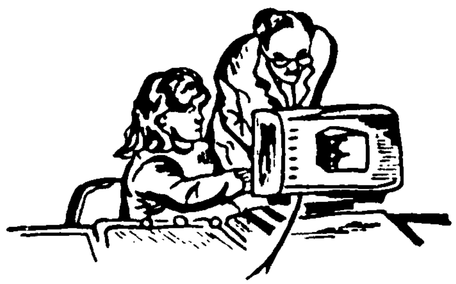
- Ada Joint Program Office (AJPO)
  - CREASE
  - Ada Information Clearinghouse
  - ASEET
- Commission of the European Communities (CEC)
  - Ada Training Needs Assessment

Training Focus

- Largely targets practitioners
- Primarily focussed on Ada as a language

Need for Training

Ada and Software Engineering Training for All Key Populations



060887TR4A

13

The Solutions

- Longer Term:
  - Develop appropriate training
- Near Term:
  - Select the best from existing offerings

The Ada Training Guide

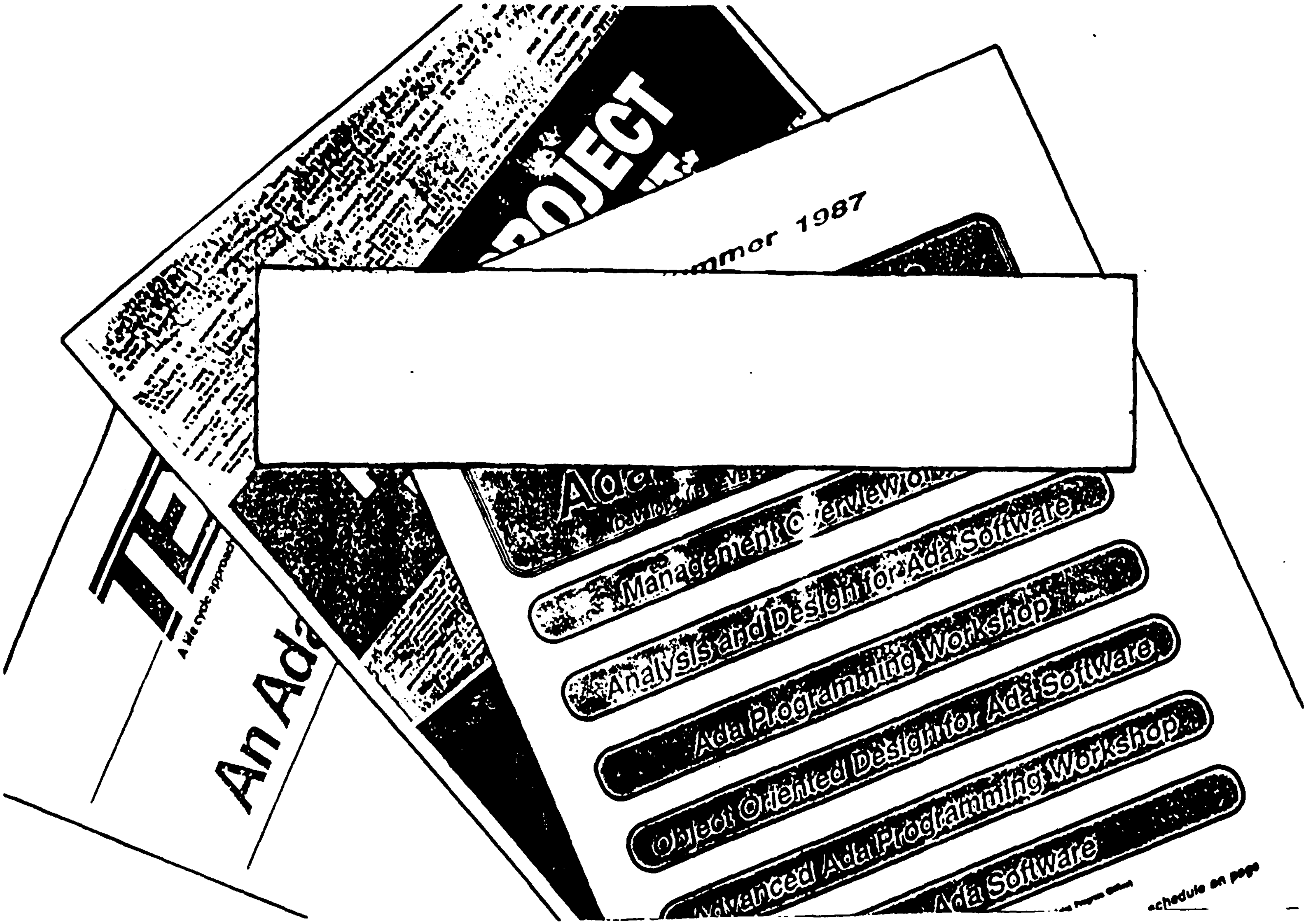
- Four sections:
  - procedures for training needs assessments
  - approaches to selecting training
  - training evaluation procedures
  - Ada insertion strategies

Needs Assessment Procedures

- Procedures
  - Content
  - Context

Ada Training Selection

060887TR8A


PROJECT
mmer 1987
Management Overview
Analysis and Design for Ada Software
Ada Programming Workshop
Ada Programming Workshop
Object Oriented Design for Ada Software
Object Oriented Ada Programming Workshop
Advanced Ada Programming Workshop
Ada Software
schedule on page
An Ada
A life cycle approach

Approaches to Selecting Training:

Evaluating the options. . .

060887TR9A

Ada Insertion Strategies

- Training goes hand in hand with
  - Management education
  - Standards revision
  - New/revised tools
  - Environment adaptation
  - Revised reward systems

The Ada Training Guide

- Design philosophy
  - brief
  - prototyped first
  - tested
- Status
  - interdisciplinary team
  - prototyped with Program Office personnel

The Ada Training Guide
Prototype Target:

060887TR12A

Open Questions

- Is our test group typical?
- Do we need one guide or a set of guides?
- Must the guide be domain-specific?
- How effective can the guide be?

Ada®

FROM A MANAGEMENT PERSPECTIVE

MAJOR CHARLES ENGLE: UNITED STATES MILTARY ACADEMY
WEST POINT, N.Y.

1LT ANTHONY DOMINICE: KEESLER TECH TRAINING C NTER
KEESLER AFB, MS.

SPONSORED BY:

Ada JOINT PROGRAM OFFICE (AJPO)

Ada SOFTWARE ENGINEERING EDUCATION AND TRAINING (ASEET) TEAM

OVERVIEW

* Rationale for development
* Capabilities and advantages
* Life Cycle application

CHARACTERISTICS OF DoD SOFTWARE

* Expensive
* Incorrect
* Unreliable
* Difficult to predict
* Unmaintainable
* Not reusable

WHAT YOU MAY HAVE HEARD ABOUT Ada

* It's a cure—all for DoD computing
* It's just another D--- acronym
* It's a programming language
* It's "just another programming language"
* Life cycle costs, support environments, STARS, Methodologies, SEI ?? !! It's everything

WHAT YOU NEED TO HEAR ABOUT Ada

Plain and simple ...

* Ada is a standardized computer programming language developed by the DoD for use in embedded computer systems

* Ada is the BEST tool available for meeting the software engineering requirements of the DoD

THE CRITICALITY OF SOFTWARE

* Hardware is no longer the dominant factor in the hardware/software relationship
  - Cost
  - Technology
* The demand for software is rising exponentially
* The cost of software is rising exponentially
* Software maintenance is the dominant software activity
* Systems are getting more complex
* Life and property are dependent on software

FACTORS AFFECTING DoD SOFTWARE

* Ignorance of life cycle implications
* Lack of standards
* Lack of methodologies
* Inadequate support tools
* Management
* Software professionals

TRADITIONAL APPROACH TO SOFTWARE

* A necessary evil

* A black art

* Guru's and magicians in a dark room

THE FUNDAMENTAL PROBLEM

* Our inability to manage the COMPLEXITY of our software systems

* Lack of a disciplined, engineering approach

SOFTWARE ENGINEERING

THE ESTABLISHMENT AND APPLICATION OF SOUND ENGINEERING =>

* Environments
* Tools
* Methodologies
* Models
* Principles
* Concepts

SOFTWARE ENGINEERING

COMBINED WITH =>

* Standards

* Guidelines

* Practices

SOFTWARE ENGINEERING

TO SUPPORT COMPUTING WHICH IS =>

* Understandable
* Efficient
* Reliable and safe
* Modifiable
* Correct

THROUGHOUT THE LIFE CYCLE OF A SYSTEM

(C. McKAY, 1985)

PROGRAMMING LANGUAGES
AND
SOFTWARE ENGINEERING

* A programming language is a software engineering tool

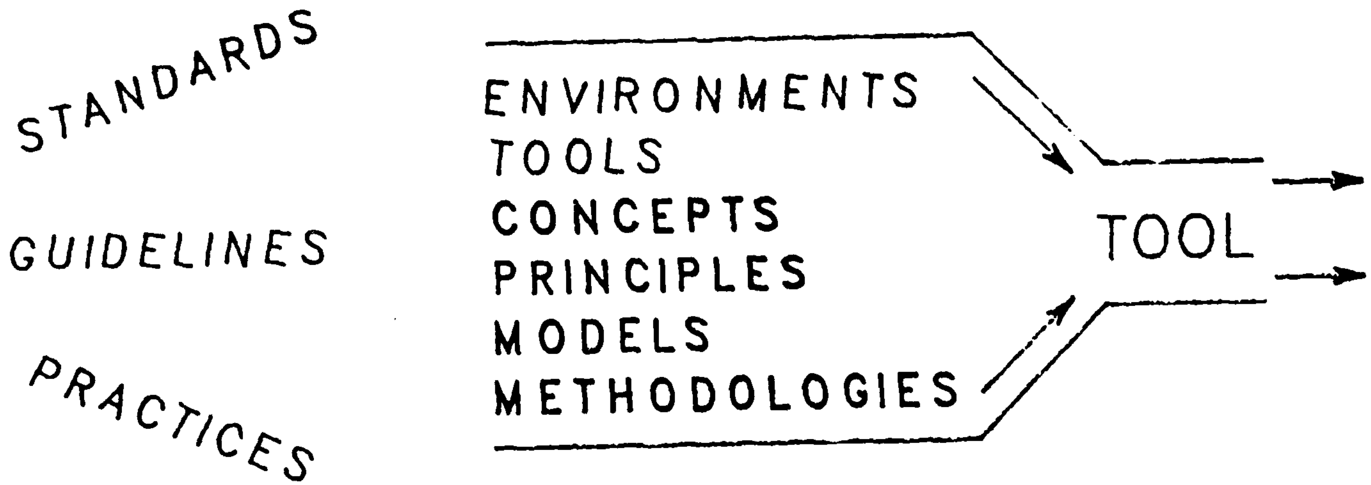
* A programming language EXPRESSES and EXECUTES design methodologies

* The quality of a programming language for software engineering is determined by how well it supports a design methodology and its underlying models, principles, and concepts

TRADITIONAL PROGRAMMING LANGUAGES
AND
SOFTWARE ENGINEERING

Programming Languages

* Were not engineered
* Have lacked the ability to express good software engineering
* Have acted to constrain software engineering



Ada
AND
SOFTWARE ENGINEERING

Ada

* Was itself "engineered" to support software engineering
* Embodies the same concepts, principles, and models to support methodologies
* Is the best tool (programming language) for software engineering currently available

	ENVIRONMENTS →	T →
STANDARDS	TOOLS	
GUIDELINES	CONCEPTS	O
	PRINCIPLES	O
PRACTICES	MODELS	
	METHODOLOGIES →	L →

PRINCIPLES OF SOFTWARE ENGINEERING

* Abstraction
* Modularity
* Localization
* Information hiding
* Completeness
* Confirmability
* Uniformity

MAJOR FEATURES OF Ada

* Standardization
* Readability
* Program Units
* Separate Compilation
* Subprograms
* Packages
* Strong Typing
* Typing Structures
* Data Abstraction
* Tasks
* Exceptions
* Generics

MAJOR FEATURES OF Ada

* Standardization
* Readability
* Program Units
* Separate Compilation
* Subprograms
* Packages
* Strong Typing
* Typing Structures
* Data Abstraction
* Tasks
* Exceptions
* Generics

SYSTEMS ENGINEERING

* Analyze problem

* Break into solvable parts

* Implement parts

* Test parts

* Integrate parts to form total system

* Test total system

REQUIREMENTS FOR EFFECTIVE SYSTEMS ENGINEERING

* Ability to express architecture

* Ability to define and enforce interfaces

* Ability to create independent components

* Ability to separate architecture issues from implementation issues

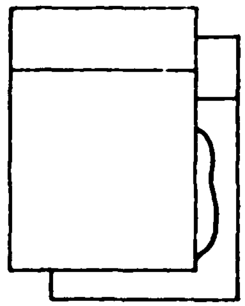
PROGRAM UNITS

* Components of Ada which together form a working Ada software system

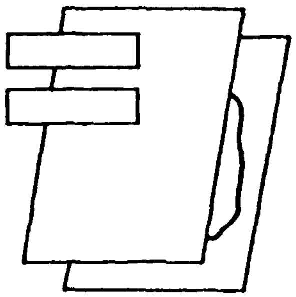
* Express the architecture of a system

* Define and enforce interfaces

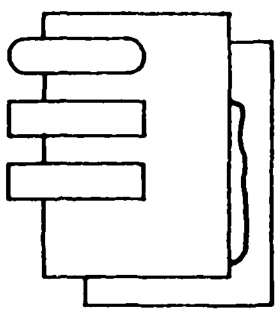
PROGRAM UNITS



SUBPROGRAMS — Working components that perform some action



TASKS — Performs actions in parallel with other program units

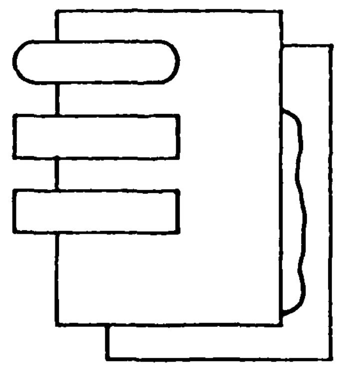


PACKAGES — A mechanism for collecting entities together into logical units

PROGRAM UNITS

* Consist of two parts: specification and body

SPECIFICATION: Defines the interface between the program unit and other program units (the **WHAT**)



BODY: Defines the implementation of the program unit (the **HOW**)

PROGRAM UNITS

* The specification of the program unit is the only means of connecting program units

* The interface is enforced

* The body of a program unit is not accessible to other program units

* There is a clear distinction between architecture and implementation

COMPARING DESIGNS:

A METHODOLOGY FOR TEACHING SOFTWARE ENGINEERING

SECOND ANNUAL ASEET SYMPOSIUM

JUNE 1987

PUTNAM P. TEXEL

TEXEL & COMPANY

48

ABSTRACT

PAPER DESCRIBES:

- SUCCESSFUL PEDIGOGICAL TECHNIQUE
- USE IN TRAINING SOFTWARE ENGINEERS
- "PORTED" TO TRAIN TECHNICAL MANAGEMENT

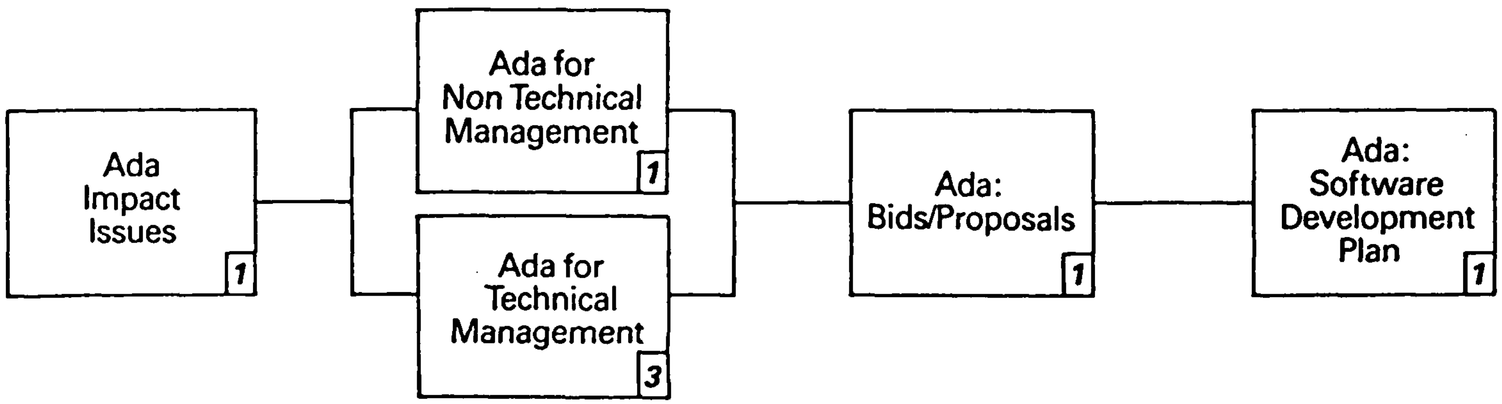
HISTORY

UNDER CONTRACT TO GENERAL DYNAMICS TO:

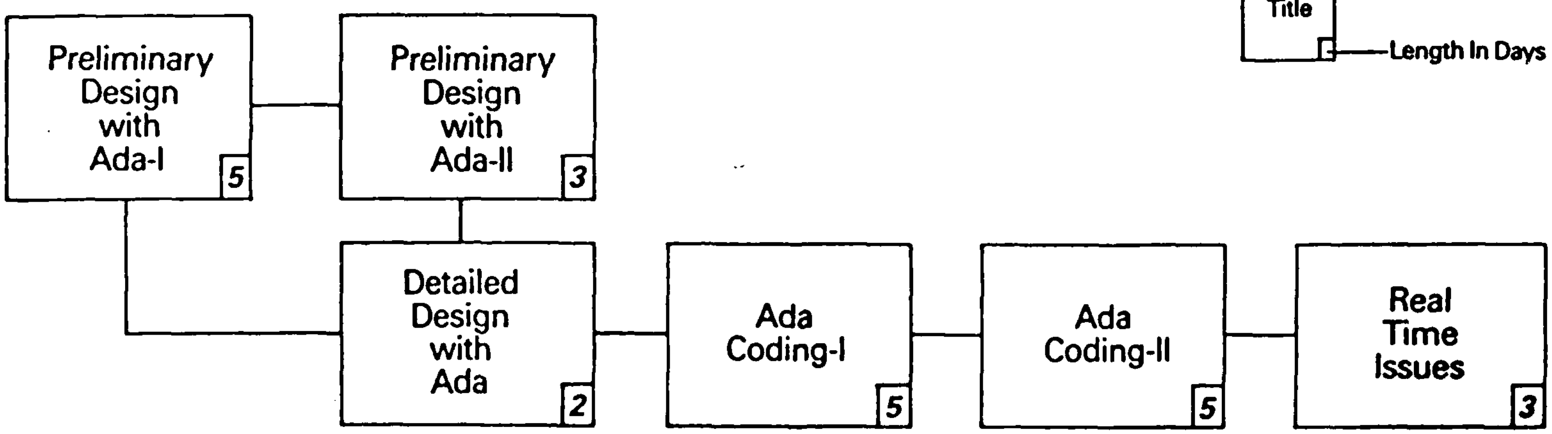
- PROVIDE SOFTWARE ENGINEERING CURRICULUM BASED ON Ada
- IMPLEMENT COURSES
- TEST TEACH COURSES
- TRAIN TRAINERS

An Ada® Training Series

MANAGEMENT TRACK



SOFTWARE ENGINEERING TRACK



LEGEND:

Title — Length In Days

TEXEL & Co.

ASEET 87 - 3

®Ada is a registered trademark of the DOD (OUSDRE).

51

CODING SEQUENCE IMPLEMENTED FIRST
(AT GENERAL DYNAMICS REQUEST)

- ONGOING TEAM EXERCISE BASED ON ADVENTURE GAME*

* originally conceived by Dick Bolz

PRELIMINARY DESIGN

- DURING DEVELOPMENT OF COURSE IT BECAME APPARENT THAT JUST TALKING ABOUT SOFTWARE ENGINEERING AND HOW Ada SUPPORTS SOFTWARE ENGINEERING IS NOT ENOUGH

- BUILT IN SEVERAL WORKSHOPS, WITH THE FIRST ONE BASED ON:

  - VERY POOR DESIGN FROM THE CODING CLASS
  - TWO BETTER DESIGNS FROM THE CODING CLASS
  - FINAL DESIGN (LEVELS OF ABSTRACTION)

WORKSHOP MECHANICS

1. CLASS DIVIDED INTO GROUPS OF THREE - FOUR
2. STATEMENT OF REQUIREMENTS IS DISTRIBUTED TO EACH GROUP
3. FIRST DESIGN IS DISTRIBUTED
4. EACH GROUP WORKS INDEPENDENTLY (SEPARATE WORK AREAS ARE REQUIRED) AND EVALUATES DESIGN AGAINST STATEMENT OF REQUIRMENTS
5. GROUPS RETURN IN ONE HOUR AND PRESENT THEIR FINDINGS TO ENTIRE CLASS
6. SEQUENCE IS REPEATED FOR SECOND - FOURTH DESIGNS

!! NOT EVERY BELL & WHISTLE IS REQUIRED !!

GOAL IS TO FOCUS ON LEARNING (IN THE GUT!)
SOFTWARE ENGINEERING

STATEMENT OF REQUIREMENTS

ROOMS - minimum of five (5)

- maximum of six (6) exits per room
- points collected upon first entrance to room
- initial message associated with each room (not a long & a short message) displayed only the first time

TREASURES - minimum of four (4) treasures

- points collected when pick up treasure
- decremented when drop treasure

COMMANDS

- drop items
- pick up items
- move
- display

WIN

- accumulating most points
- aquire certain prize

Ada from the Trenches : A Classroom Experience

Jaime Niño
Computer Science Department
University of New Orleans

Introduction.

In August of 1984, the department of Computer Science of the University of New Orleans (**UNO**) started **implement**ing the programming language **Ada** as **the departamental language**.

This paper is a personal report of this 3 year effort.

Structure of lecture.

University Setting.

Departamental Setting

Curriculum overview.

Ada as a Primary Programming Language.

Ada curriculum implementation history.

Ada teaching experience.

Student Population Response.

Compiler Experience.

Conclusion.

University Setting.

UNO is a **public urban university** of approximately 17,000 students . The second largest of the institutions governed by the Louisiana State Board of Regents.

UNO is on a **semester system**, with most Computer Science courses having three hours of lecture per week, and offering three credits to the student.

UNO computer resources :

a cluster of **four VAX 8600** 16 Mb computers running VMS Version 4.5.

DECNET

150 Zenith microcomputers with Winchester discs

numerous terminals throughout campus.

campus-wide **Ethernet.**

Departament Setting.

We offer the **baccalaureate** in Computer Science

master degree in Mathematics with specialization in C S.

The department consists of **10 full time faculty** members .

Supported by part time faculty, several teaching assistants and paper graders.

400 Computer Science majors at **UNO,**

1500 students studying Computer Science courses at any one time.

120 declared Computer Science majors studying CSCI-1583, (CS1)

90 students enrolled in CSCI-2120 (or CS 2 in the ACM curriculum),

60 in CSCI-2125 (Data Structures or CS 7).

Curriculum Overview.

CSCI-1583

Prerequisites : Plane trigonometry with Algebra

Corequisites : Calculus I or Discrete Mathematics.

Syllabus overview:

Computers in general: computer systems organization, basic computer organization and history of computers.

Programming priciples : programming languages concepts, examples of programming languages, typical programming tasks, software lifecycle, software quality, algorithm design (top-down, bottom-up) with strong concentration on top-down design and step-wise refinement. Structure Programming, Abstract data types, algorithm testing, documentation.

Ada: primitive types, data manipulation via typed objects, control statements, subprograms, scope and visibility, records with fixed fields, constrained arrays, use of packages. Attributes and Input/Output using the standard input and output files.

CSCI-2120

Prerequistes : CSCI-1583 and either 1) credit in Discrete Mathematics or 2) concurrent registration in Discrete Mathematics and credit in Calculus I.

Syllabus overview.

Software life cycle. Abtract Data Types. Encapsulation, localization and Information Hiding. Design Techniques : Top Down design. Bottom Up Design. Separate compilation and top-down coding. Top-down testing. Other design techniques. Ada Block structure. Scope and visibility. Recursion and backtracking. Programming in the large : Bottom-up design and packages. Robustness : . Testing (Structured walk-throughs) and verification. Program assertions, loop invariants. Partial program correctness.

Ada Review. Unconstrained arrays, records with discriminants.Packages.Files.Exceptions.

CSCI-2125

Prerequisites : CSCI-2120 , Discrete Mathematics and Calculus I.

Syllabus overview.

Abstract Data Types : Specification, design, validation , implementation.

Study of typical data structures as ADT's : Stacks, Queues, Lists, Recursive Types, Binary trees, General trees, Graphs. ADT's and algorithms : searching, sorting, hashing. Other ADTS's.

Ada : Generic Packages. Access Types. Variant Records.

Ada as a primary programming language. (WHY Ada)

In [Evans et al 1985] reasons against Ada :

complexity
size
lack of compilers
lack textbooks.

Prevealing reason : **complexity.**

In the course of the implementations we have shown that we can find a **suitable subset of sequential Ada** that does not do a disservice to the language and that supports and serves us well in our teaching goals.

Teaching goals: teach and stress the principles and fundamentals of programming to produce readable, maintainable and correct programs.

Principles : structure programming, modularization, information hiding, data abstraction, encapsulation and localization.

Goals : produce a program that is correct, reliable, robust, maintainable, verifiable and portable. Teach the principles and techniques available from software engineering

Ada and goals:

Standard

Modern

Commercial

Ada was designed to support all of these principles and the needs of modern software development.

Students get the opportunity to put the principles taught in class to practice.

The advanced features found in Ada to support the development of embedded systems has given us the opportunity to be able to use Ada in the upper level courses which have a programming component.

Ada has unified our curriculum

Use Ada as a tool to illustrate programming techniques and principles. Learning of Ada per se is not a goal. What this means is that we teach the necessary Ada syntax and semantics to illustrate programming principles and techniques.

Ada is a very rich and complex language whose wealth of features can overwhelm any well experienced programmer. We had to carefully trim Ada to make it into a manageable language from the point of view of the teaching of programming.

Ada curriculum implementation history

Fall of 84 : CSCI-1583 (Ada). CSCI-1060 (Pascal)

Spring of 85: two sections of CSCI-1583 and one section of CSCI-2120 using Ada. One section of CSCI-1060 and one section of CSCI-2120 using Pascal.

Fall of 85 CSCI-1583, CSCI-2120 and CSCI-2125 using Ada. One section of CSCI-2120 and CSCI-2125 using Pascal

Spring of 1986 the only section remaining to be faced out from our curriculum was a section of CSCI-2125 using Pascal.

Fall of 1986 we taught all the programming core course sections of CSCI-1583, 2120, 2125 using Ada. There were no sections of any of those courses offered using Pascal in that semester or afterwards. We continue teaching an introductory course using Pascal which is aimed to the Liberal Art students as well as an introductory course using FORTRAN aimed to the Engineering School students.

By fall 1986 faculty teaching higher courses with a programming component could expect to have many of the students in the course with Ada experience.

Ada Teaching Experience.

CSCI-1583 : teaching of the elements of structured programming, data abstraction, algorithm development and the necessary syntax to support such .

In 1583 the students get exposed to the following concepts:

1. Structure programming.
2. Abstract Data Types.
3. Top-down design.

Ada has a complete and fully braketted control structure set.

Fuctions do not take modifiable parameters.

Ada distinguises between OUT and IN OUT parameters in contrast with Pascal and its by-reference parameter mode.

Having given the definition of abstract data types, we can illustrate it via the Ada primitive types and the fact that no implicit coercion is allowed in Ada.

We can also illustrate it by introducing derives types of primitive types.

Students can be given the opportunity of using non-primitive types via packages provided by the instructor.

With this simple instance, an instructor can illustrate Abstract Data Types, team programming as well as the information hiding principle.

Top down design is supported in Ada via **subprograms** and **separate compilation** and this last Ada feature gives the instructor an oportunity to give the student the experience of programming in the large students write simple drivers for packages. Use separate compilation to give the opportunity to students of being part of the writing of a rather "interesting" program by assigning the students the writing of a subprogram which will be linked to a main subprogram written by the instructor. This feature can be exploited further by given different group of students different subprograms to write. Notice that in these situations the students can be made part of a programming effort with a minimum knowledge of Ada syntax. To use packages students need to know how to declare and give value to the arguments how to use the conditional and while control structures, how to call subprograms and to include packages with their drivers.

The actual writing of subprograms can be helped by giving simple but well defined operations to implement, whose logic is simple and do not need the use of sophisticated Ada features.

CSCI-2120 teach to the student the principles underlying the production of correct, portable, maintainable, modifiable and verifiable programs.

In the teaching of the second course the choice of Ada is more rewarding. Among the main concepts taught in this course we have:

1. Structure Programming revisited.
2. Design methodologies . (Top-down , bottom-up, among others).
3. Robust programming and error trapping.
4. Recursion and Backtracking.
5. Abstract Data Types revisited.

The concept of abstraction which was taught using subprograms can be reenforced using unconstrained arrays and discriminated records and user defined enumeration types. Using separate compilation student can get experience in top-down design , top-down coding and top-dow testing. With packages students learn bottom-up design and coding. Exceptions simplifies the introduction and the actual implementation of error trapping.

CSCI-2125 specification, design and implementation of Abstract Data Types. At this point packages, private and limited types will support the concepts of information hiding, encapsulation and data abstraction.

In all courses we teach Ada that supports principles and concepts.

I teach arrays slices to be used to make calls to subprograms with unconstrained array type parameters; this is an example of the implementation and use of abstration. I do not teach all the possible ways to form aggregate expressions. I do not see a principle that can be illustrated with this activity.

Student Population Response.

The student population consists mostly of commuter students who live in the metropolitan area. The great majority of them work while attending classes. difficulty of a given programming course lies in the subject matter and not in the programming language.

The questions and difficulties students bring to my office are within the same class of questions and difficulties as when I was teaching Pascal. I have yet to meet a student who attributes his or her difficulties directly to the complexity of the language.

The attrition rate using Ada in the core programming courses was not affected. For the first course we had experience up to 50% attrition rate, for the second course up to 33% attrition rate and for the third course no more than a 25% attrition rate when Pascal was the departamental programming language. Many students drop the class due to the fact that they do not have enough time to devote to programming. As I mentioned above, most of the students are full time and hold a job while going to school. We feel that when each student owns or has access to a personal computer the attrition rate will decrease significantly.

Compiler Experience.

In the fall of 84 we started the teaching of Ada using a pre-validation version 1.3 of the Telesoft Ada compiler. This was not a full Ada compiler.

Notorious features lacking in that compiler version that were noticed in the teaching of the first programming class were the lack of generic io packages of text_io, the lack of some type transfer functions, and the fact that output parameters could be read.

In January of 1985, the Telesoft-certified release, Version 2.1 was substituted. This version for VMS was submitted for DoD validation early in 1984 but not validated, failing two programs in the test suite.

With some minor difficulties, this version was used for the spring semester of 1985 in both the first and the second programming courses.

During the spring of 1985, the DEC Ada compiler became available. We have used that compiler from the summer of 1985. This is a full Ada compiler which is supported by the DEC VAX/VMS symbolic debugger. This is an excellent compiler. It generates relatively small object files and the run time of the executable image is more than adecuate for an academic environment.

We currently have the latest validated version 3.10 of the Telesoft Ada compiler. Judging from the list of problems we have encountered with this version (see appendix), it is clear that the compiler to use in a VAX/VMS environment is the DEC Ada; Telesoft Ada is an adecuate choice, but requires large amounts of secondary space to be allocated to each user

Conclusion.

We feel that the choice of Ada as a departamental programming language has been benefitial to the teaching of programming by having a language that is both modern and supports the needs of modern software development, for the unification of the curriculum courses with a programming component and ultimately for the student who has the opportinity of exposure to a language that is making a definite impact in our field. We have shown that it is possible to use Ada in the introductory courses with more benefits than disadvantages; and these benefits accrue as the students take the more advanced courses using Ada. It is benefitial to the teaching of programming to use a programming language that is a standard language, with modern features and which is a real language that is making a definite impact in the world.

Introducing Ada® and Its Environments into a Graduate Curriculum

by

Major Patricia K. Lawlis

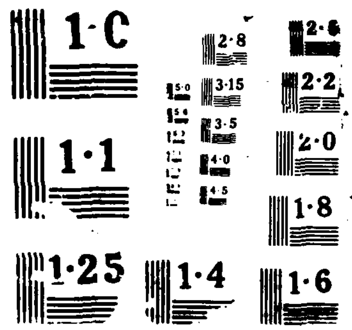
Karyl A. Adams

Air Force Institute of Technology (AFIT)

lawlis%asu@csnet-relay

AD-A189 650 ANNUAL ASEET (ADA SOFTWARE ENGINEERING EDUCATION AND TRAINING) SYMPOSIUM (SLIDES) HELD IN DALLAS TX ON 9-11 JUNE 1987(U) ADA JOINT PROGRAM OFFICE ARLINGTON VA 2/4

UNCLASSIFIED 11 JUN 87 F/G 12/5 NL


1.0
2.8
2.5
3.15
2.2
3.5
2.0
1.1
4.0
1.8
1.25
1.4
1.6

Overview

- The early years
- The impact of compilers
- A changing curriculum
- Growth of the Ada Program
- AFIT's current curriculum
- The value of Ada research
- Ada's future at AFIT

The Early Years

- Started introductory Ada course around 1980
- Course ran once per year
- Programming assignments were "hand compiled"
- Ada infiltrated the compiler courses
- Masters thesis research began
- Difficult but promising years

The Impact of Compilers

- First was unvalidated Telesoft in 1984
- Validated Verdix and DEC systems in 1985
- Set the stage for a curriculum change

A Changing Curriculum

- Ada replaced a combination of Pascal and C
- Redesigned courses focused more on s/w engineering
- Started with class entering June 1985
- Impacted faculty in 2 departments

Growth of the Ada Program

- 60 students in 3 classes
  - 40 used same Verdix system
- Started with 3 Ada literate professors
- Other professors gained interest out of necessity
- Other courses began to use Ada for implementation
  - Compiler
  - Graphics
- Acceptance was beginning

AFIT's Current Curriculum

- 2 versions of introductory Ada course
- Courses developed especially for Ada
  - Software environments
  - Real-time programming
- Other courses gradually converting
  - Compiler
  - Graphics
  - Data structures
  - Operating systems
  - Database

The Value of Ada Research

- Masters thesis work since 1981
  - Some of most significant referenced in paper
- Participation in the Ada community feeds research
  - ASEET (Ada S/W Engineering Education & Training)
  - E & V (APSE Evaluation & Validation)
- Sets up a "circle of influence" which impacts curriculum
  - Environments class started a prototype APSE
  - ARCADE provides support for theses and classes

Ada's Future at AFIT

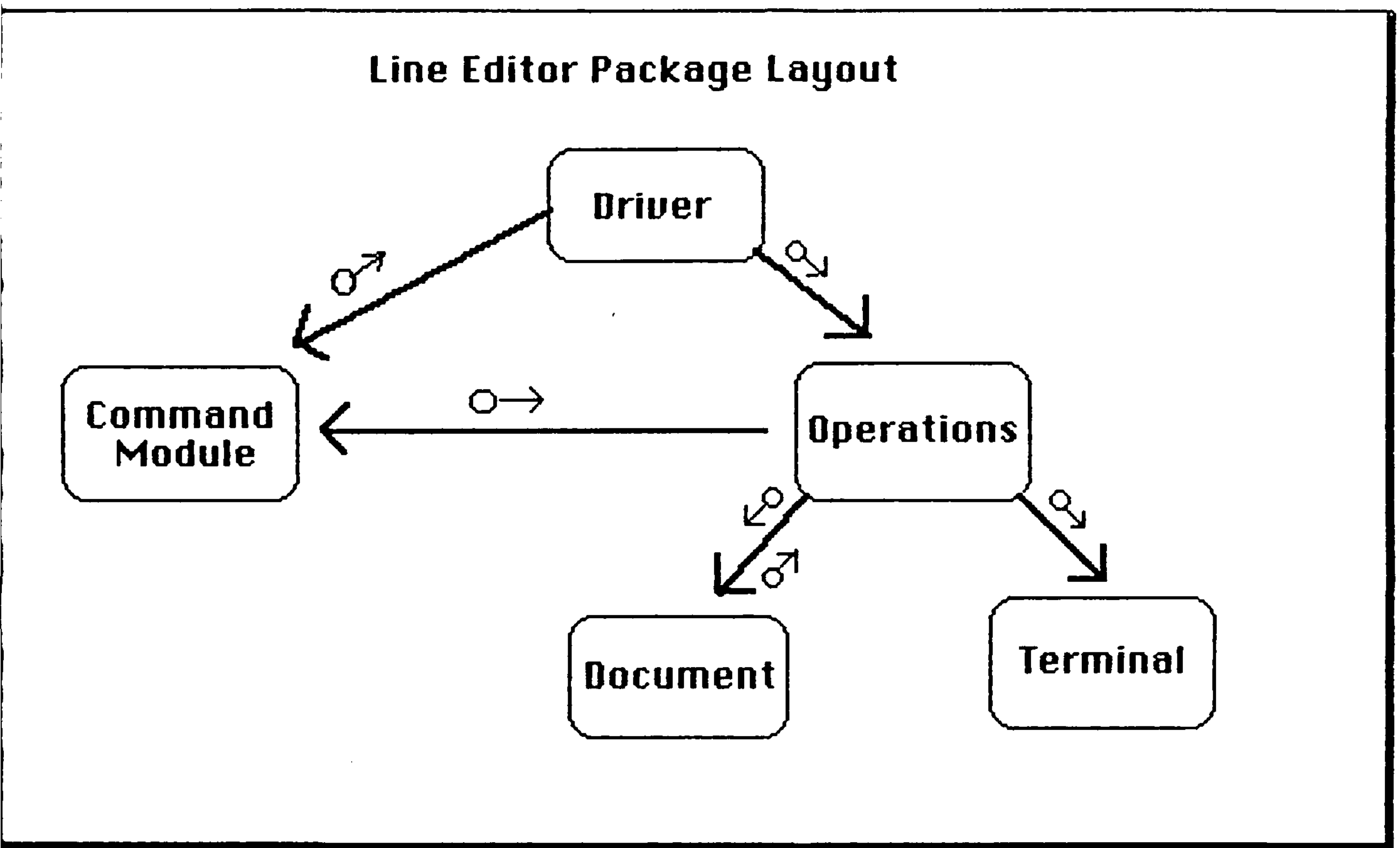
- Curriculum growth seemed slow, but it happened
- Commitment is now established
  - Needs to be nurtured/maintained
- Continues to require faculty attention
- Faculty participation in the community is important

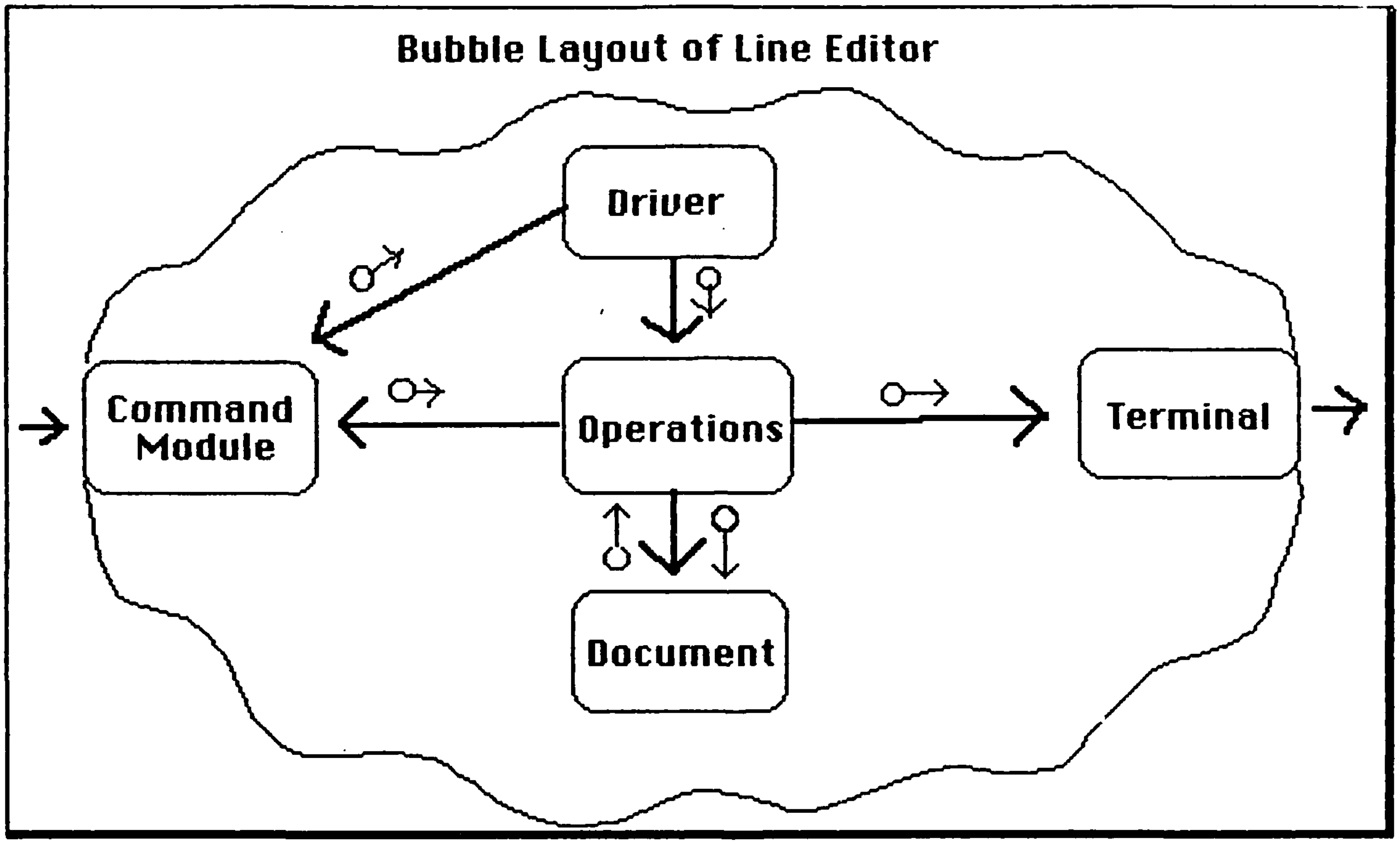
Formal Specification Techniques
in an Ada-based
Software Engineering Course

Charlene M. Hamiwka and Laurence J. Latour
Department of Computer Science
University of Maine

```
package STACK_PACKAGE is

   type STACK is private;

   procedure PUSH (I: in INTEGER;
                   S: in out STACK);
   procedure POP (S: in out STACK);
   procedure TOP (I: out INTEGER;
                  S: in STACK);

private
   type STACK is {implementation dependent};
end STACK_PACKAGE;
```

```
package STACK_PACKAGE is

   type STACK is private;

   procedure PUSH (I: in INTEGER;
                   S: in out STACK);
   procedure POP (S: in out STACK);
   procedure TOP (I: out INTEGER;
                  S: in STACK);

-- Legality
--        For all T, &(T)
-- Equivalences:
--        0 < n < 124 ==>
--             PUSH^n(a_i).POP = PUSH^(n-1)(a_i)
--             PUSH(a).PUSH^124(a_i) = PUSH^124(a_i)
--             T.TOP = T
--        n > 0 ==>
--             POP^n.PUSH (a) = PUSH  (a)
-- Values:
--        V(T.PUSH(a).TOP) = a mod 255

private
   type STACK is {implementation dependent};
end STACK_PACKAGE;
```


Line Editor Package Layout
Driver
Command
Module
Operations
Document
Terminal


Bubble Layout of Line Editor
Driver
Command Module
Operations
Terminal
Document

A Student Project to Extend Object-Oriented Design

Richard F. Vidale
Boston University
Boston, Massachusetts 02215

Charlene R. Hayden
GTE Government Systems Corporation
Needham, Massachusetts 02194

Today's Talk

Background:

- Courses and Student Projects in Ada and Software Engineering at Boston University
- The Space Station Command & Control Project (MITRE Project)

How we Extended Object-Oriented Design:

- The Adaptive Routing Algorithm Project (GTE Project)
- What We've Learned

Courses and Student Projects

Semester	Courses	Student Projects
Fall '83	SC 465 - System Design SC 511 - Software Engineering	
Spring '84	EK 215 - Introduction to Ada	MITRE project
Fall '84	EK 215 - Introduction to Ada SC 511 - Software Engineering SC 465 - System Design	
Spring '85	EK 215 - Introduction to Ada	
Fall '85	EK 215 - Introduction to Ada SC 511 - Software Engineering SC 465 - System Design	Draper project
Spring '86	EK 215 - Introduction to Ada	GTE project
Fall '86	EK 215 - Introduction to Ada SC 511 - Software Engineering SC 465 - System Design	
Spring '87	EK 215 - *Introduction to Ada*	*Ada*GRAPH project

Compilers used in Courses at Boston University

Semester	Courses	Compiler	Students
Fall '83	SC 465 - System Design	NYU Ada/Ed	95
	SC 511 - Software Engineering	AAEC Pascal	40
Spring '84	EK 215 - Introduction to Ada	NYU Ada/Ed	29
Fall '84	EK 215 - Introduction to Ada	NYU Ada/Ed	20
	SC 511 - Software Engineering	AAEC Pascal	44
	SC 465 - System Design	NYU Ada/Ed, DG/Rolm	91
Spring '85	EK 215 - Introduction to Ada	DG/Rolm	34
Fall '85	EK 215 - Introduction to Ada	NYU Ada/Ed	20
	SC 511 - Software Engineering	AAEC Pascal	28
	SC 465 - System Design	DG/Rolm	78
Spring '86	EK 215 - Introduction to Ada	DG/Rolm	32
Fall '86	EK 215 - Introduction to Ada	NYU Ada/Ed	20
	SC 511 - Software Engineering	AAEC Pascal	50
	SC 465 - System Design	DG/Rolm, Symbolics	87
Spring '87	EK 215 - Introduction to Ada	DG/Rolm	26

Space Station Command & Control Program

Requirements:

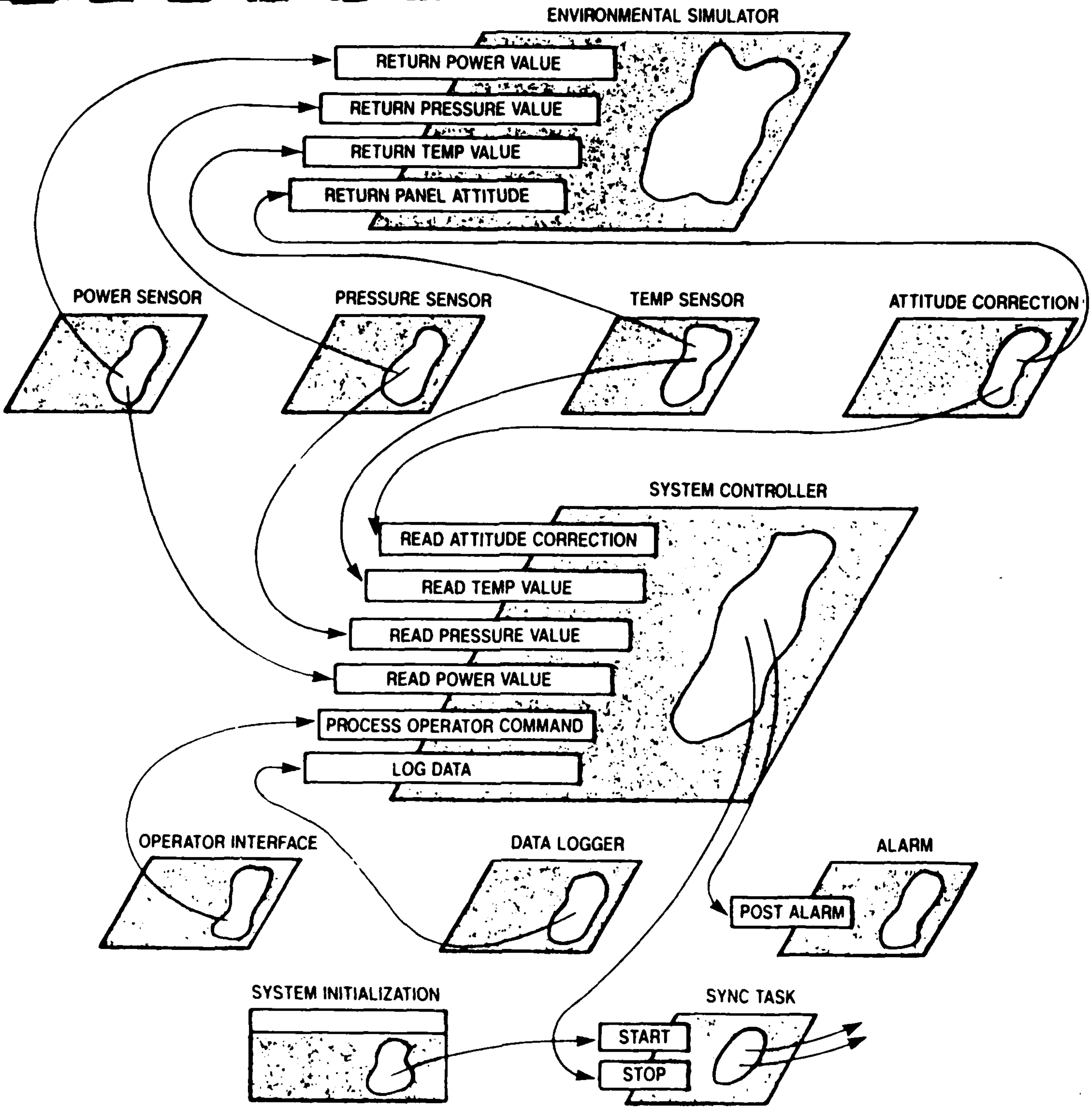
- Monitor space station sensors
- Calculate attitude coordinates for solar panel of space station
- Operator-defined times of sensor checking
- Initialize sensor threshold limits
- Detect out-of-bounds limits and send alarm to operator console
- Alarm message includes sensor, value, & time of day

Requirements (Cont'd):

- Operator control for:
  - Selectively enabling/disabling sensors
  - Selectively recording sensor values, alarm conditions, and current solar panel attitude values
  - Selectively changing changing the sensor threshold limits
  - *Terminating the program*
- When more than one sensor is scheduled to be sampled at the same time, the following priorities apply:
  - Priority 1 -- Space station power
  - Priority 2 -- Space station pressure
  - Priority 3 -- Space station temperature

Abbott/Booch Object-Oriented Design

1. Define the problem (software requirements)
2. Develop an informal strategy (English description)
3. Formalize the strategy
   - (a) Identify objects and their attributes
   - (b) Identify operations on the objects
   - (c) Establish the interfaces
   - (d) Implement the operations


ENVIRONMENTAL SIMULATOR
RETURN POWER VALUE
RETURN PRESSURE VALUE
RETURN TEMP VALUE
RETURN PANEL ATTITUDE
POWER SENSOR
PRESSURE SENSOR
TEMP SENSOR
ATTITUDE CORRECTION
SYSTEM CONTROLLER
READ ATTITUDE CORRECTION
READ TEMP VALUE
READ PRESSURE VALUE
READ POWER VALUE
PROCESS OPERATOR COMMAND
LOG DATA
OPERATOR INTERFACE
DATA LOGGER
ALARM
POST ALARM
SYSTEM INITIALIZATION
SYNC TASK
START
STOP

Disadvantages of Object-Oriented Design

- Does not account for all the software modules needed.
- Diagrams do not show data flow.
- Not well-suited for large, hierarchical systems.
- Does not provide means of analyzing task timing and synchronization.

Adaptive Routing Algorithm Program

Requirements:

- Develop program for a multiprocessor within one node of a data switching network
- Each node maintains tables of:
  - Distances (number of hops to reach a node)
  - Minimum delay time
  - Message routing

Requirements (cont'd):

- In parallel, compute minimum delay times to neighbors and update distance and minimum delay time of each neighbor
- When a link is broken or established, a parallel process corrects the distance and minimum delay times
- Number of nodes, neighbors of nodes, and periodic update interval are constants

Extended Buhr Design Methodology

1. Identify objects in the problem domain. Represent them as "clouds" and show data flow between objects. Threads of control can be shown as chronologically numbered data flows.

2. Develop scenarios of object interaction using preliminary timing diagrams.

3. Define global Ada data types for inter-object data flow. Compile global data types package(s).

4. Represent problem-domain objects as Ada program units.

5. Draw Buhr-style structure graph of program architecture. Compile specifications of program units.

6. Add call directions to timing diagrams.

7. Generalize task sequencing requirements from timing diagrams and encode them in Task Sequencing Language (TSL) specifications.

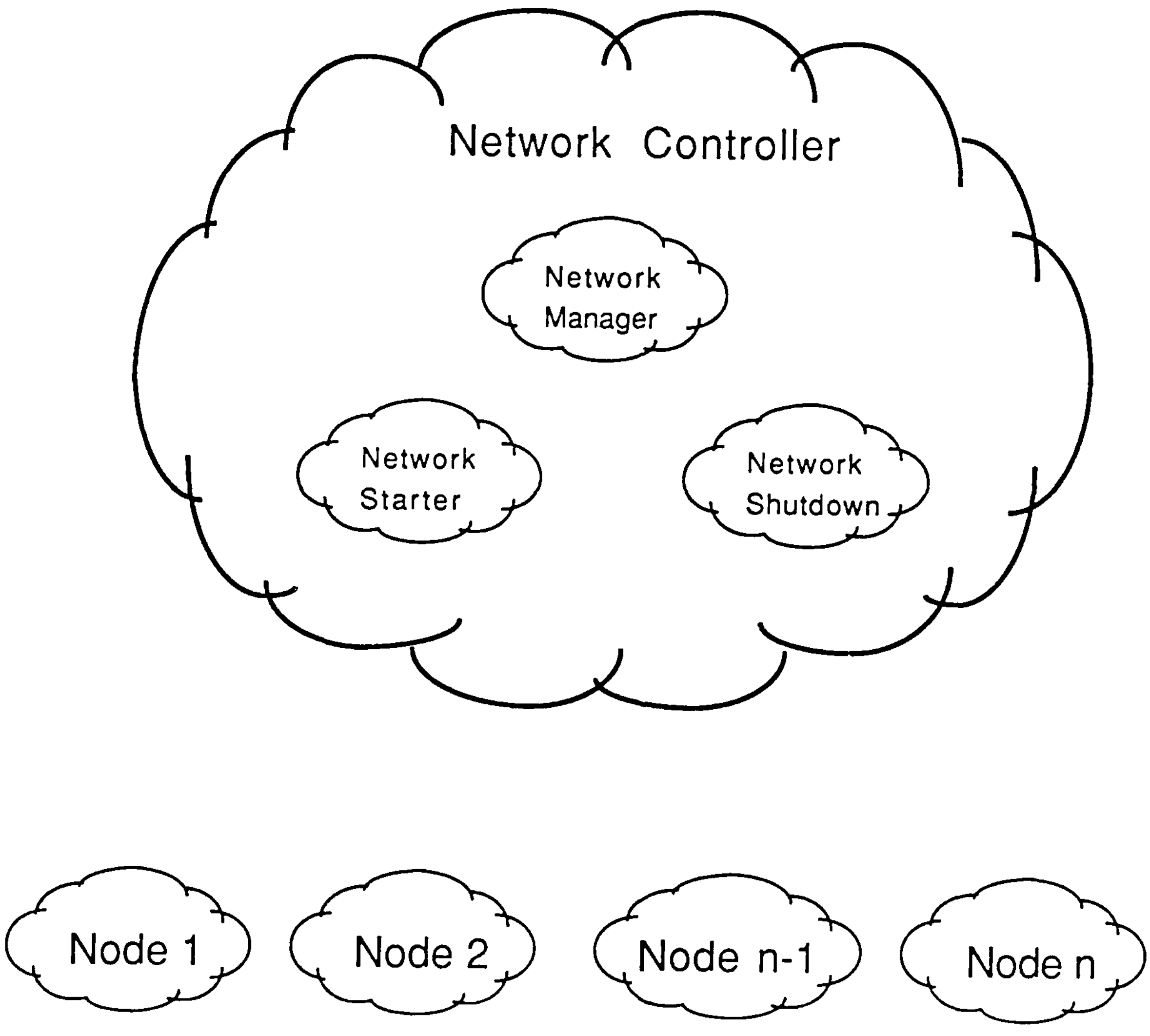
8. Draw Petri nets to describe local and global task sequencing.

9. Write control skeletons for program unit bodies.

10. Walk through Petri nets to verify the control skeletons.

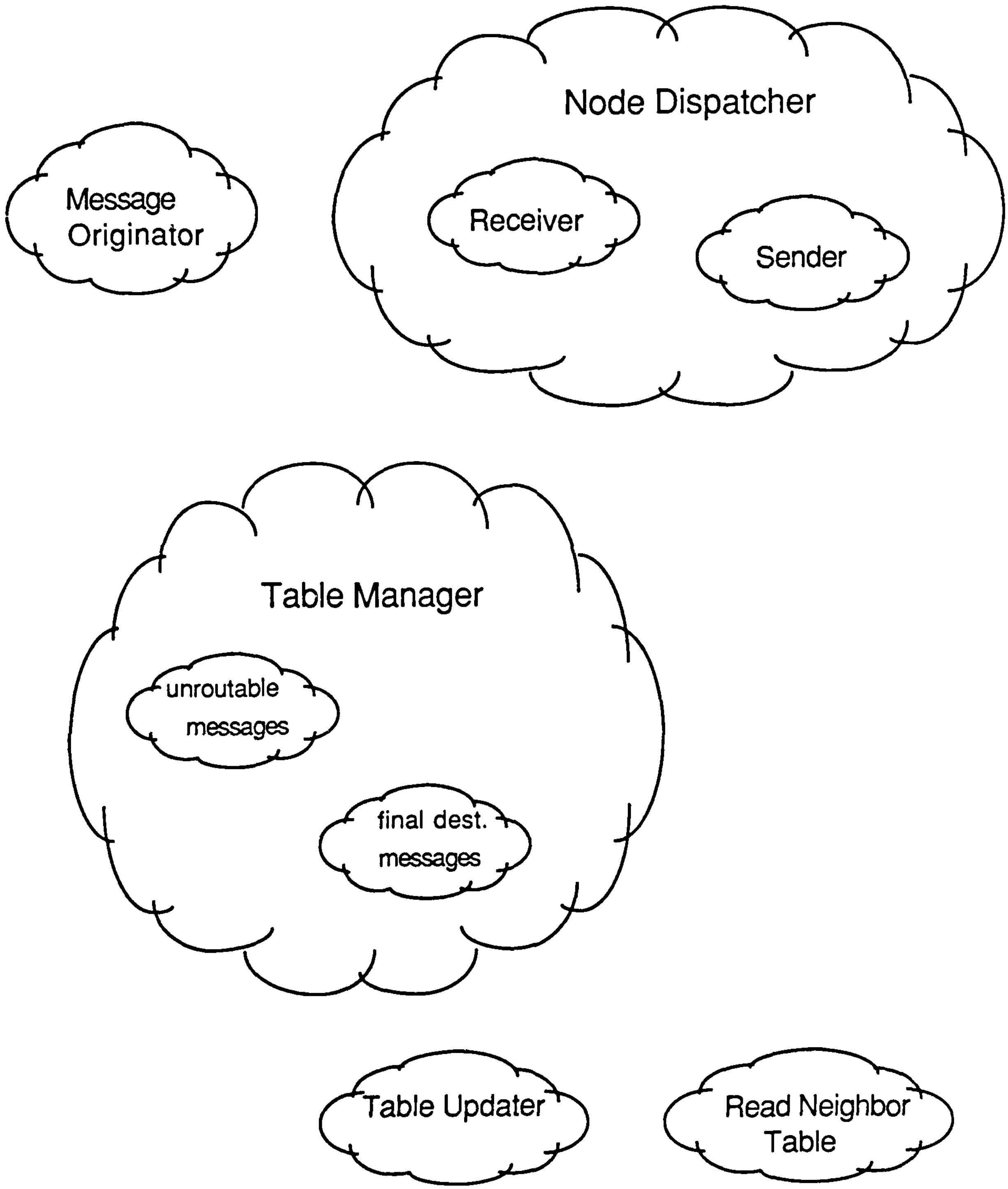
Network Level Cloud Diagram

Diagram 1

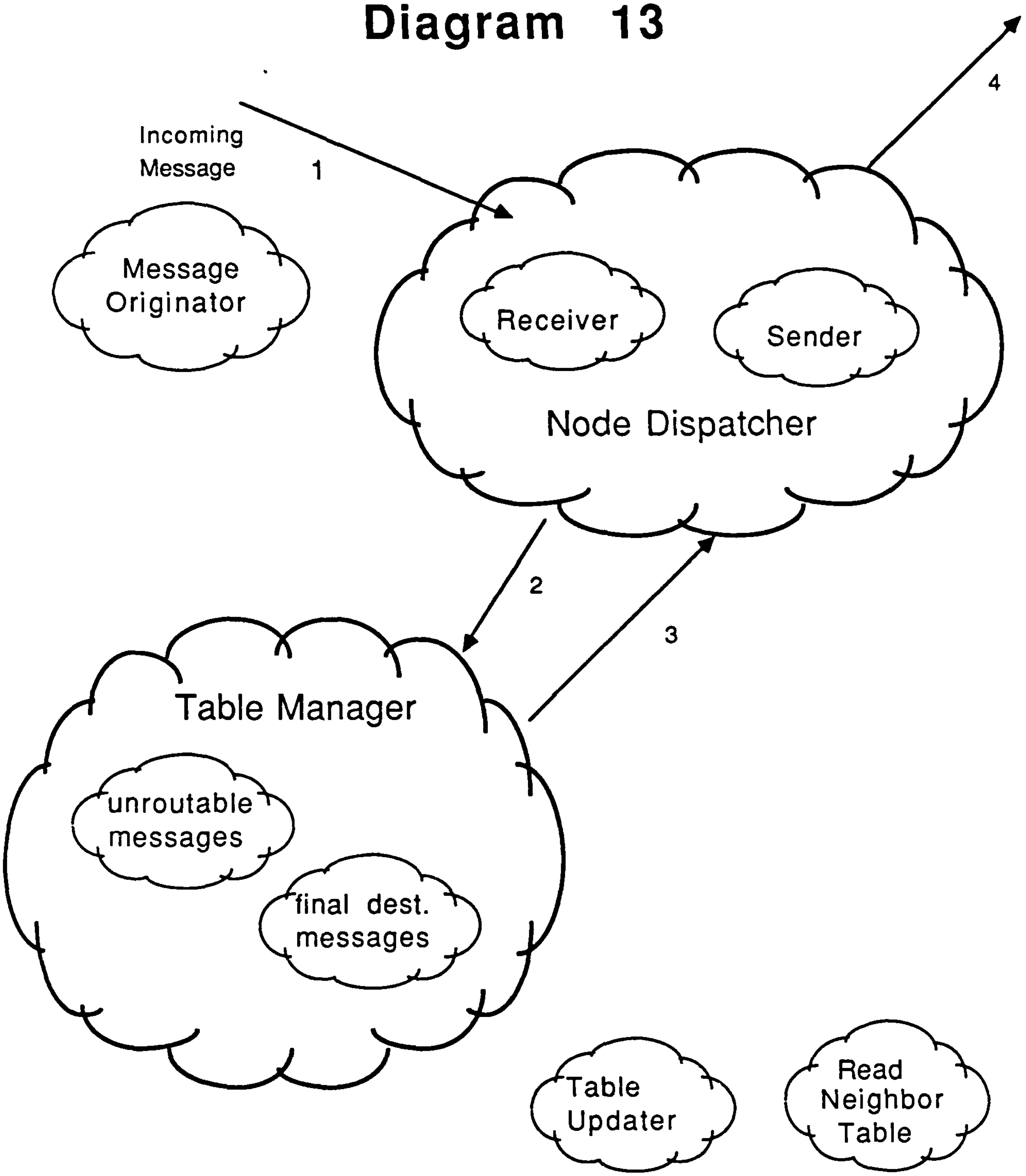


Node Level Cloud Diagram

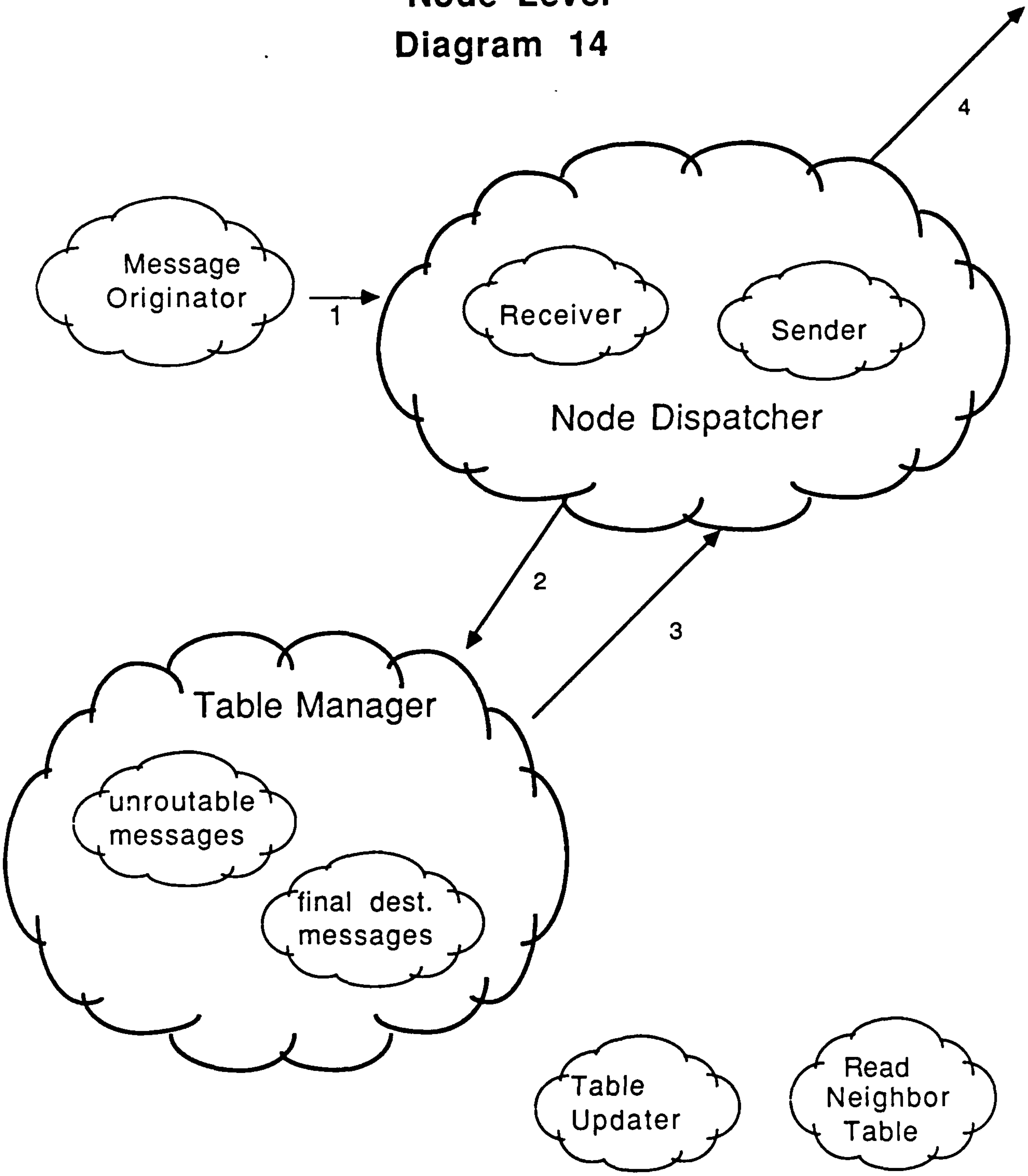
Diagram 3



Send User Message-Node Level
Diagram 13

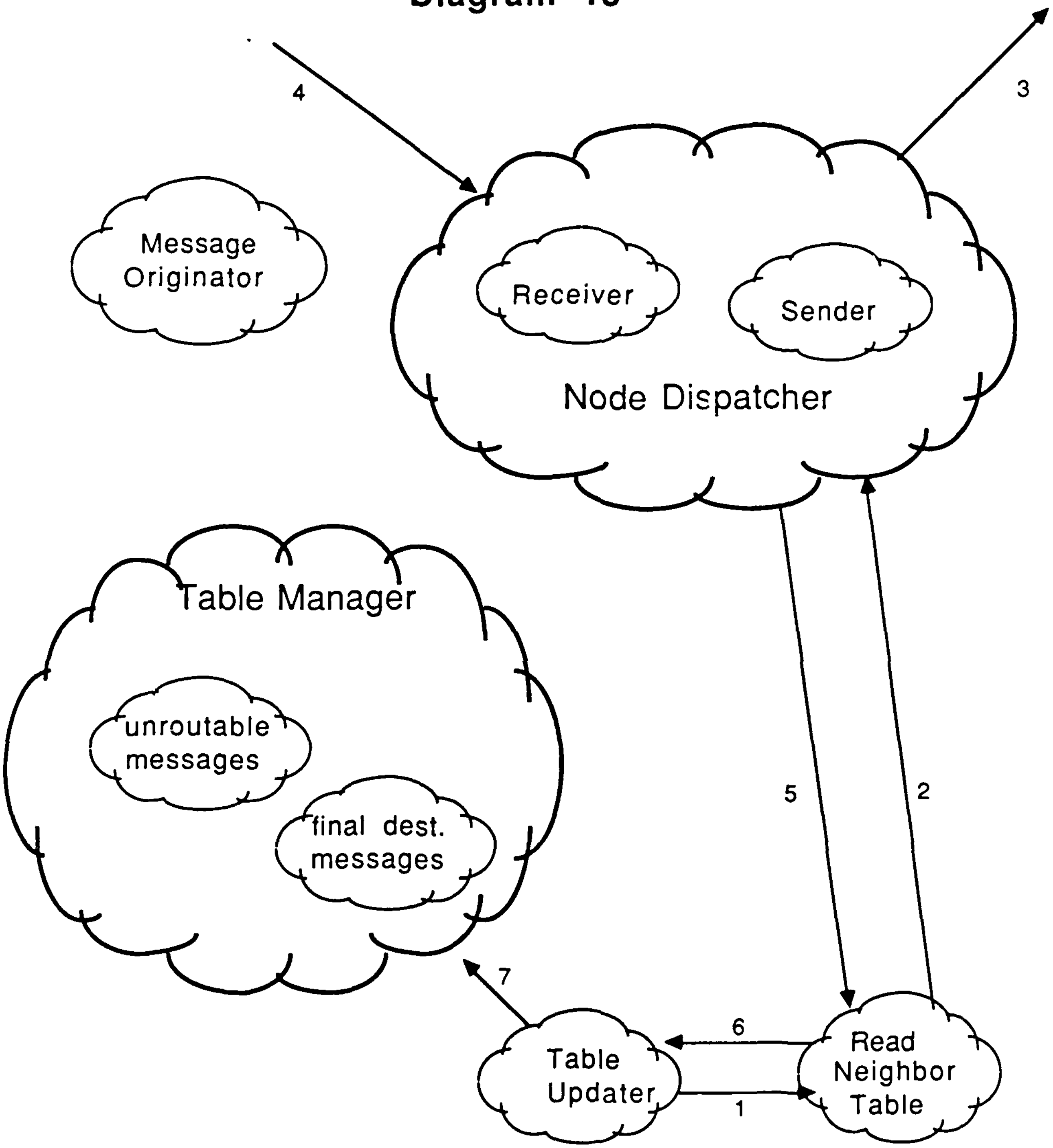


Send User Message (Originate Message) -Node Level
Diagram 14

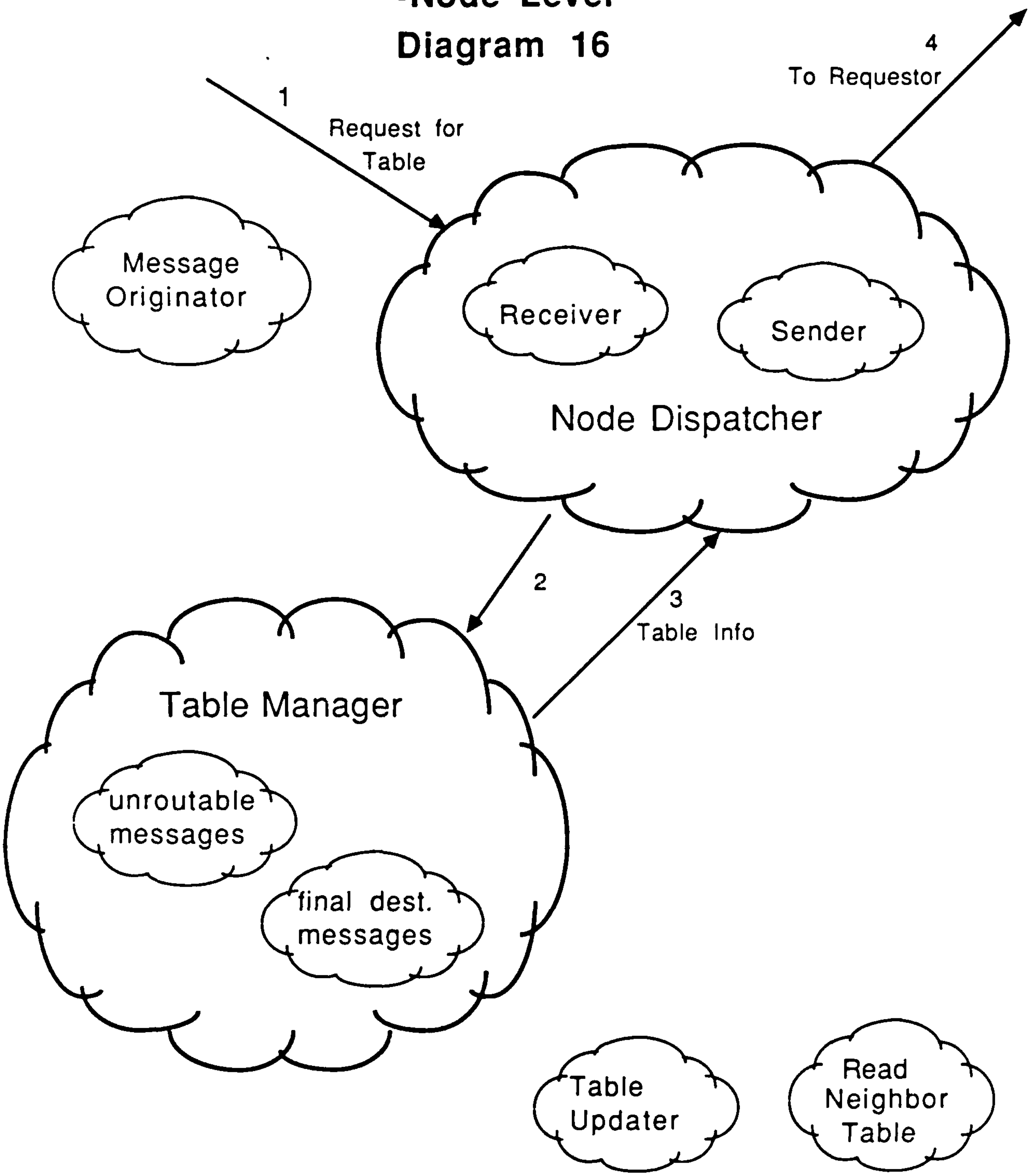


Read Neighbor Table (Initiate)-Node Level

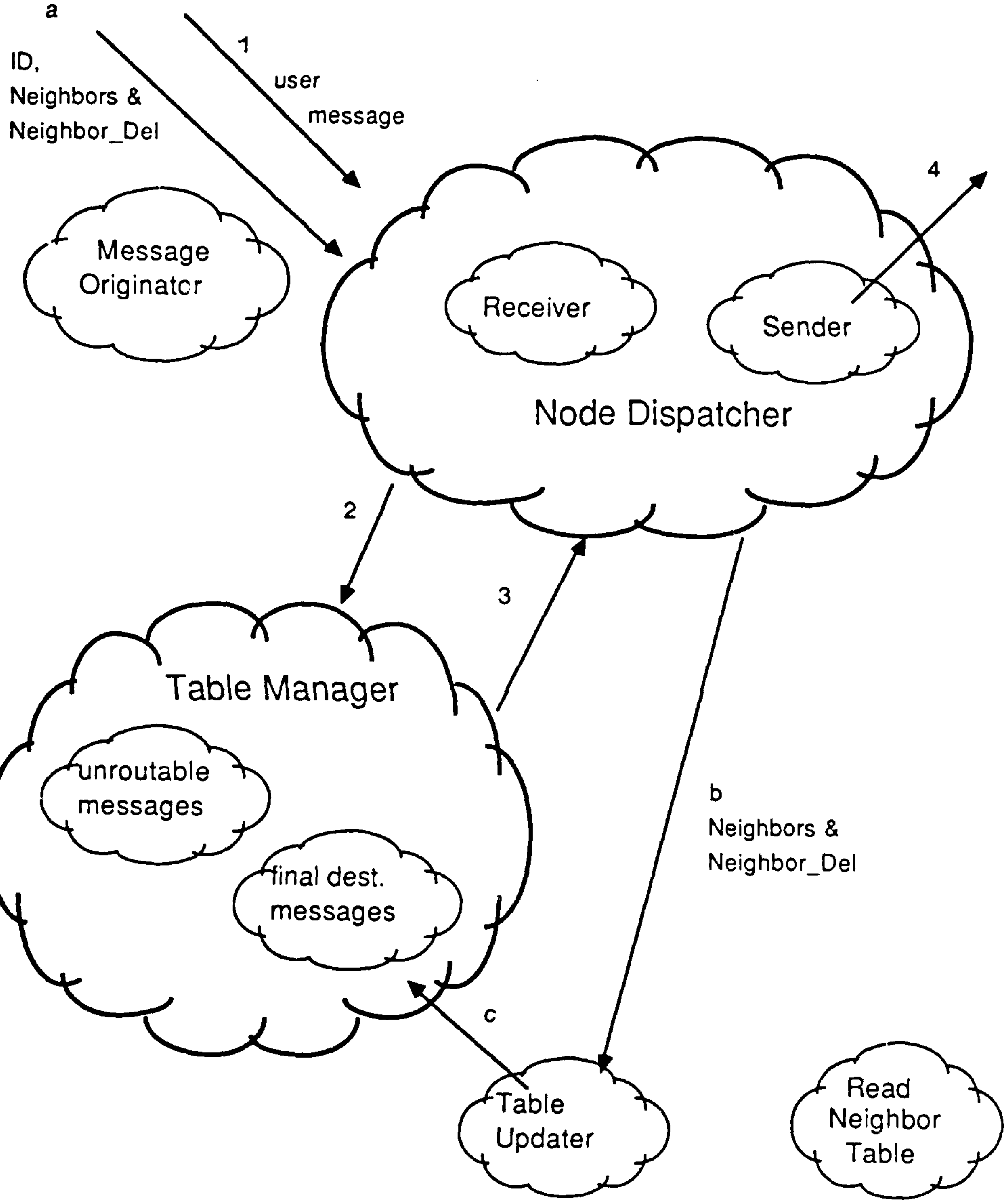
Diagram 15



Read Neighbor Table (Read Request) -Node Level Diagram 16

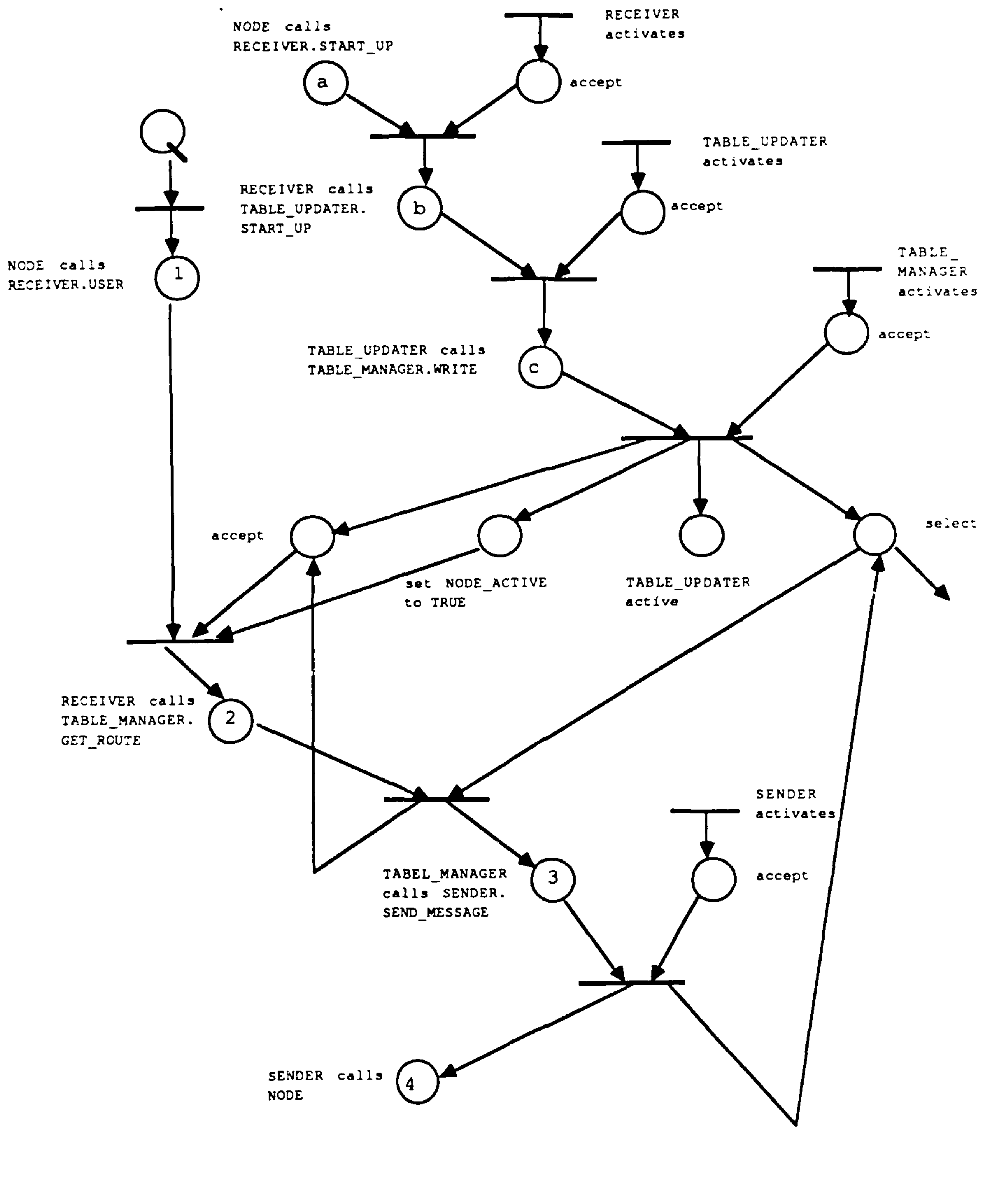


System Startup vs. User Messages
Diagram 17



STARTUP VS USER MESSAGES

Diagram 29



Control Skeletons

Diagram 34

```
task body RECEIVER:

begin
    accept START_UP
    while RECEIVER_ACTIVATED loop
        select
            accept BREAK
        or
            accept RESTORE
        or
            accept USER
        or
            accept READ_TABLE_REQUEST
        or
            accept READ_TABLE_RESPONSE
        else
                -- Check the shut down conditions.
                -- If node_active is false and there is not message
                -- being queued out the change the RECEIVER_ACTIVATED
                -- flag to false.
        end select
    end loop
end RECEIVER


task body TABLE_MANAGER:

begin
  while TABLE_MANAGER_ACTIVATED loop
    select
      accept GET_ROUTE do
        NODE_DISPATCHER.SENDER.SEND_MSG
      end GET_ROUTE;
    or
      accept READ do
        NETWORK_NODE(DESTINATION).NODE_RECEIVE
      end READ;
    or
      accept WRITE
    else
      if INITIALLY_ACTIVATED and not(NODE_ACTIVE) and
         (GET_ROUTE'COUNT = 0) and (READ'COUNT = 0) and
         (WRITE'COUNT = 0) then
         -- set TABLE_MANAGER_ACTIVATED flag to false
      end if
    end select
  end loop
end TABLE_MANAGER
```

Conclusions from GTE Project

- EBDM forced early analysis of concurrent threads of control. Later phases went smoothly because of the understanding gained during preliminary design.

- More training in the EBDM and its constituent techniques would have helped.

- The most valued techniques in EBDM were

  - Cloud diagrams

  - Buhr diagrams

  - Control skeletons

- The next most valued techniques were
  - Petri nets
  - Task Sequencing Language
- The least valued technique was Timing Diagrams
- The monitor was helpful in the testing phase
- The style guidelines *made for easy reading of another's* code

What We've Learned

About Teaching Ada:

- Slow, buggy compilers turn off students to Ada
- The general areas of difficulty are:
  - Tasking
  - Generics
  - Program architecture

- It's hard to appreciate Ada's features:
  - Strong typing
  - Private types
  - I/O packages
  - Model numbers
- Many students are put off by the difficulties
- But some become zealots

About Teaching Software Engineering:

- A substantial project is essential
- A formal testing phase reveals the benefits of good software design
- Teamwork and organization determine success
- The course is well received by most students

About Multi-Tasking Ada Projects:

- The prerequisites for success are:
  - A talented team
  - Prior software design team experience
  - An adequate design methodology
  - An introduction to Ada, with tasking
  - An adequate Ada development environment

What's Ahead

- Encore Ada compiler in place this fall
- A new course, SC 525, this fall in embedded computer software design, using Ada
- Phase out SC 465 in two years, to be replaced by the EK 215, SC 525 sequence
- Acquire dedicated target machines

An Evolution in Ada Education for Academic Faculty

M. Susan Richman, Ph.D.
Director
Ada Education and Software Development Center
The Pennsylvania State University at Harrisburg
Middletown, PA 17057   tel: (717) 948-6082

	COURSE ONE	COURSE TWO	COURSE THREE
Length of Course	10 weeks	6 weeks	1 week
Number of Instructors	1	2(1 lect,1 lab)	1
Number of Students	15	13	8
Computer	VAX 11/780	VAX 11/780	DG MV/10000
Compiler	Ada/Ed Interpreter	Ada/Ed Interpreter	Rolm production quality compiler
Extent of Computer Support	Limited Other users	Free access Other users	Free access Other users
Pre-Course Assignment	Discouraged	Motivational readings	Readings on Pascal Subset of Ada
Post-Course Assignment	None	None	Final project
Credit vs Non-Credit	Non-credit	Non-credit	Credit
Ada Features Covered in Programming Assignments	All except Tasking Generics Low-Level IO	All except Low-Level IO	All except Low-Level IO
Daily Format	a.m. lecture p.m. lab 9:30a.m. to 4 p.m.	a.m. lecture p.m. lab 9 a.m. to 4:30 p.m.	Alternate approx. 1 hr lect/1 hr lab 8:30 a.m. to 5 p.m.

RECOMMENDATIONS for INTENSIVE COURSE

PREREQUISITES: Pascal, readings

NUMBER OF INSTRUCTORS: 2
(1 sufficient if experienced)

SYSTEM and COMPILER: production quality compiler and high level of system support

LAB EXERCISES: tailored to reinforce lectures and to exploit reusable components

TEXTS: Reference Manual is indispensable, also 1 or more texts treating Ada as a second language

GUEST LECTURERS: valuable; use if time permits

ALTERNATE MEDIA (e.g. CAI) : can provide very useful support in lengthy course; not feasible in 1-week course

DAILY FORMAT: intersperse relatively short lab and lecture periods to minimize fatigue

REQUIREMENTS: programming assignments <u>must</u> be required; exams optional

SOFTWARE ENGINEERING PRINCIPLES: <u>must</u> be included, even at the cost of sacrificing some of the detail of Ada

SLIDES

PANEL DISCUSSION: Implementing a Life Cycle Model for Software Engineering and Ada Training

WEDNESDAY, JUNE 10, 1987

SOFTWARE ENGINEERING

SOFTWARE ENGINEERING: A DEVELOPMENT PLAN

A LONG RANGE PLAN

FOR EDUCATION AND TRAINING

TO SUPPORT

SOFTWARE DEVELOPMENT ACTIVITIES

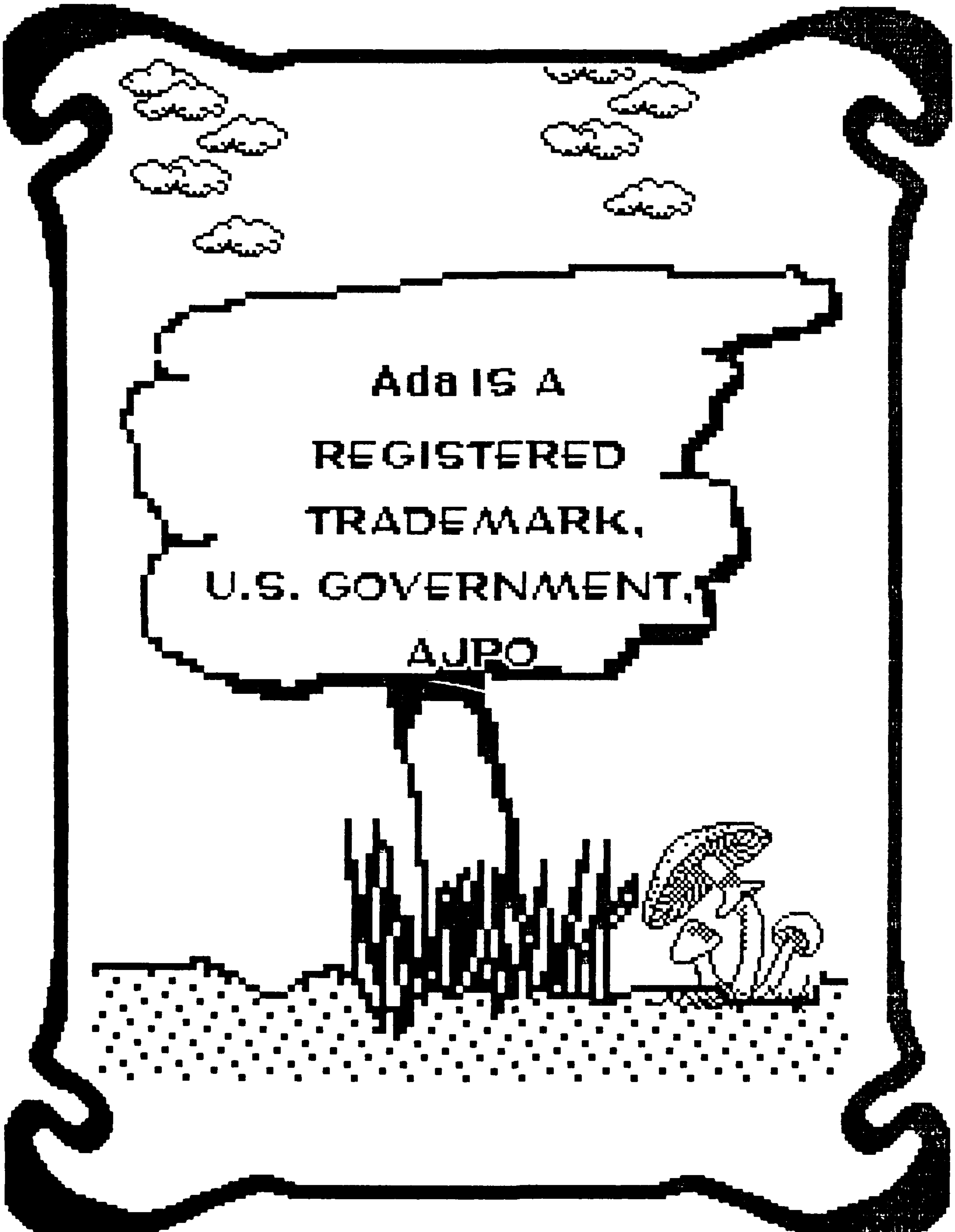
FOR THE NASA/JSC AND

UNIVERSITY OF HOUSTON-CLEAR LAKE

PRESENTED BY:

DR. GLENN B. FREEDMAN

UNIVERSITY OF HOUSTON- CLEAR LAKE


Ada IS A
REGISTERED
TRADEMARK,
U.S. GOVERNMENT,
AJPO

SOFTWARE ENGINEERING

- THE CONTEXT OF THE ISSUES
- REACTION TO MAJOR CHANGES
- FACILITATORS OR HINDRANCES TO CHANGE
- EDUCATION & TRAINING OPTIONS
- WHICH OPTION TO CHOOSE
- TRAINING CONSIDERATIONS
- SOFTWARE ENGINEERING CURRICULUM
- WHAT WILL A MODEL PROGRAM LOOK LIKE?

THE CONTEXT OF THE ISSUES

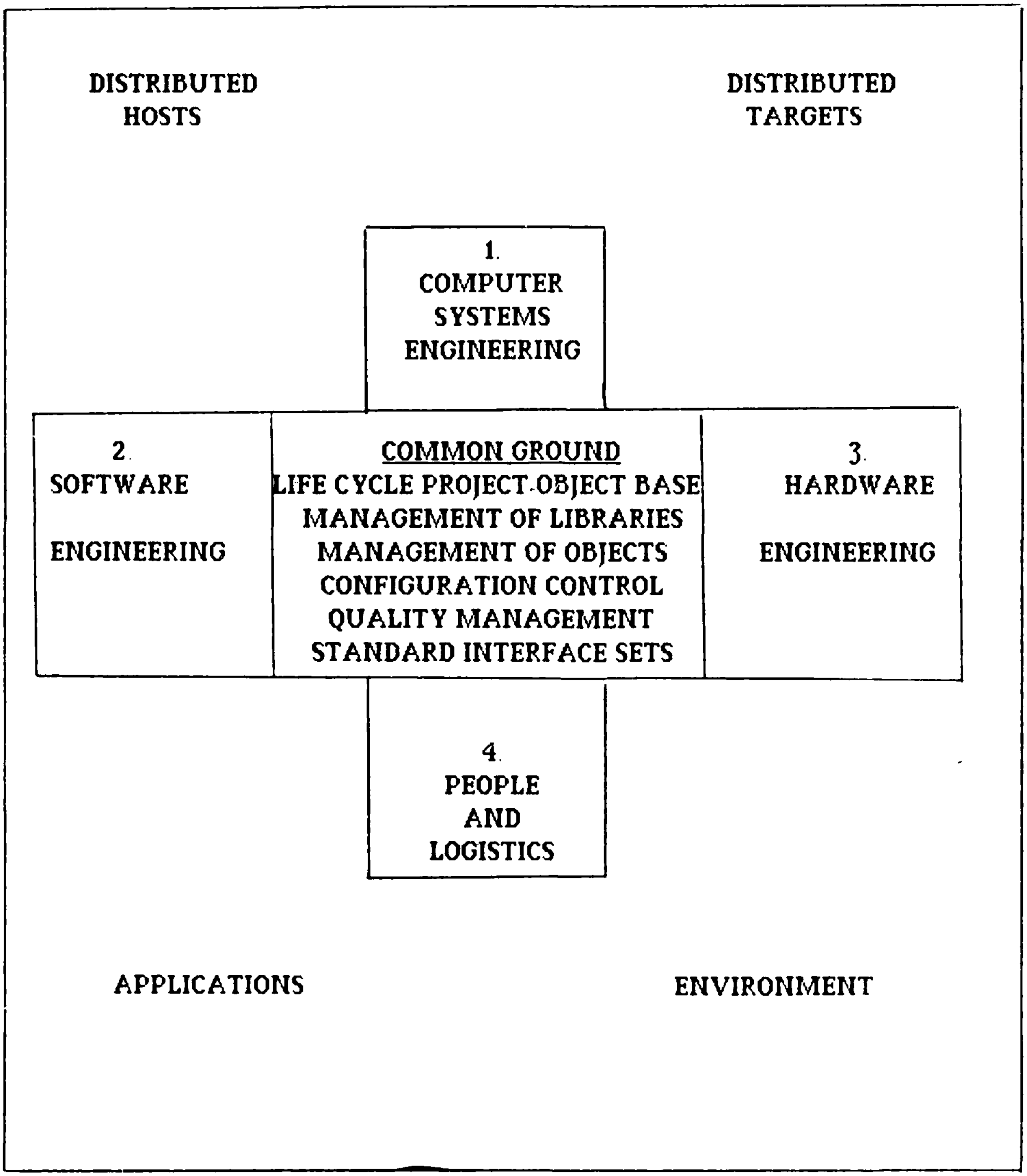
- Emergence of Software Complexity
- Computer Systems for a Space Station Environment
- Software Engineering

EMERGENCE OF SOFTWARE PROBLEMS WITH GROWTH IN SYSTEM COMPLEXITY

Attribute	1960+5 Years Programming-Any-Which-Way	1970+5 Years Programming-in-the-Small	1980+5 Years Programming-in-the-Large	1990+5 Years Program-as-Component	1990+5 Years Program-as-Deputy
Characteristic Problems	Small Programs	Algorithms and Programming	Interfaces, Management, System Structures	Integration of Heterogeneous	Incorporation of Judgment
Data Issues	Representing Structure and Symbolic Information	Data Structures and Types	Long-Lived Data Bases, Symbolic as well as Numeric	Integrated Data Bases, Physical as well as Symbolic	Knowledge Representation
Control Issues	Elementary Understanding of Control Flow	Programs Execute Once and Terminate	Program Assemblies Execute Continually	Control Over Complex Physical Systems	Programs Learn from Own Behavior
Specification Issues	Mnemonics Precise Use of Prose	Simple Input-Output Specifications	Systems with Complex Specifications	Software as Component of Heterogeneous System	Extensive Reuse of Design
State Space	State Not Well Understood Apart from Control	Small, Simple State Space	Large Structured State Space	Very Large State With Dynamic Structure and Physical Form	State Includes Development as Well as Application
Management Focus	None	Individual Effort	Team Efforts, System Lifetime Maintenance	Coordination of Integration and Interactions	Knowledge About Application Domain and Development

Source: Shaw, Mary. Beyond Programming in the Large: The Next Challenges for Software Engineering. Pittsburgh: SEI SEI Annual Report, 1985. pages 5-16.

COMPUTER SYSTEMS FOR LARGE COMPLEX ENVIRONMENTS



DEFINITION:

SOFTWARE ENGINEERING

Software Engineering is the establishment and application
of sound engineering:

- environments
- tools
- methods
- models
- principles
- concepts

combined with appropriate

- standards,
- guidelines, and
- practices

to support computing which is:

- ⇨ correct,
- ⇨ modifiable,
- ⇨ reliable and safe,
- ⇨ efficient, and
- ⇨ understandable

throughout the life cycle of the application.

(C. McKay, 1985)

REACTION TO MAJOR CHANGES

✓ The literature in social pyschology suggests that we often behave in predictable ways when facing major life changes.

✓ Changes in organizations can follow similar patterns.

✓ These stages evolve over a three to seven year period typically, with variation according to the situation.

REACTION TO MAJOR CHANGES

- Hostility and Skepticism
- Confusion and Lack of Trust
- Period of Truce; Undifferentiated Use
- Mixed Approval; Disjointed Use
- Acceptance; Initial Coordination; Differentiated Use
- Regression
- Continuing Progress; Refinement; Extension

FACTORS AFFECTING CHANGE

- Management Commitment
- Employee Attitudes
- Common Vocabularies
- Time for Changes to Occur
- Resources
- Planning
- Clarity/Viability vs.Lack/Ambiguous of Goals
- Amount of Real Work to Do
- Political Environment
- Degree of Unit Cooperation

EDUCATION & TRAINING OPTIONS

- University-style courses
- Short Courses
- Executive Summaries
- Focus Sessions/Tiger Teams
- Technical Programs/Seminars
- Conferences
- Hands-on Training
- Computer-based training
- Projects
- Apprenticeships
- Product presentations
- User Support Services
- Media Presentations
- Informal Training
- Consultants
- Reference Library and Related Services
- On-the-Job Training
- Life Experience
- Combinations of These

WHICH OPTION TO CHOOSE

- Goals
- Resources (Time, Materials, Funding)
- Knowledge vs. Skills
- Short-term vs. Long-term Learning
- Degree of Control
- Autonomy/ Motivation of Learner

TRAINING CONSIDERATIONS

- Academic Disciplines
- Prerequisite Attributes of Participants
- Environmental Attributes

Academic Disciplines

- Computer Science
- Engineering (ME, EE, CE)
- Management
- Liberal Arts

Prerequisite Attributes of Participants

- None/Intermediate/Advanced
- Same/Different/Wrong/Biased
- Excellent Attitude <--> Poor Attitude
- Long Term Goals <--> Short Term Goals

Environmental Attributes

- Commitment to Long Range Planning
- Funding Sources Identified
- Access to Appropriate Hardware & Software
- Critical Mass of Hardware & Software
- User Support Facilities
- Presence of SE Advocate
- Management Support
- Knowledge/Skills Level of Management

WHAT IS A SOFTWARE ENGINEER?

WHAT DOES THIS PERSON KNOW?

WHAT DOES THIS PERSON DO?

HOW DO WE EDUCATE SOMEONE TO BECOME A SOFTWARE ENGINEER?

HOW DO WE TRAIN SOMEONE TO BECOME A SOFTWARE ENGINEER?

SOFTWARE ENGINEERING CURRICULUM

✓ The prevailing image of the life cycle is two-dimensional, resuling in training models that are usually two dimensional.

The Clear Lake Model Curriculum has six dimensions:

- Personnel
- Activities
- Complexity and Magnitude
- Knowledge
- Environments
- Depth

SOFTWARE ENGINEERING CURRICULUM

- PREVAILING METAPHORS
- COST OF INSTRUCTION
- SAMPLE COURSE MODULES - CURRENT
- PERSONNEL CATEGORIES
- SOFTWARE ENGINEERING ACTIVITIES
- SOFTWARE ENGINEERING KNOWLEDGE
- COMPLEXITY AND MAGNITUDE
- EDUCATION VS. TRAINING
- LEVELS OF DEPTH

PREVAILING METAPHORS

- CARPENTER-ARCHITECT - BUHR
- CORPORATE MENTOR - GE
- SUPERMARKET - BOEING
- ENGINEERING/COMP.SCI./MANAGEMENT

THE COST OF INSTRUCTION

BASIC PROGRAM COST (BPC) = (EMPLOYEE TIME * 2) + UNIT COST

ANNUAL COST = BPC / USEFUL LIFE OF PROGRAM CONTENT
(IN YEARS)

PROGRAM COST PER PUPIL-HOUR =
(ANNUAL COST / HOURS OF INSTRUCTION) / PUPIL-TEACHER RATIO

EXAMPLE:

10 EMPLOYEES (AVG. $20/HR.) TAKE A 40 HR. COURSE SEQUENCE THAT WILL HELP THEM FOR TWO YEARS. THE PROGRAM COST IS $10,000.

BPC = (40*2*$20) + $10000 = $11600
AC = $11600/2 = $5800
COST/PUPIL-HOUR = ($5800*40)/10 = $145

SAMPLE COURSE MODULES - CURRENT

- Technical
- Managerial
- Support

Technical

- Component Integration
- Coding in Ada (Intro <--> Advanced)
- Methods and Tools
- Detailed Design
- Preliminary Design
- Software Requirements Analysis
- Systems Requirements Analysis
- Validation and Verification
- Configuration Management
- Documentation
- Library and Object Base Management
- Maintenance and Operations

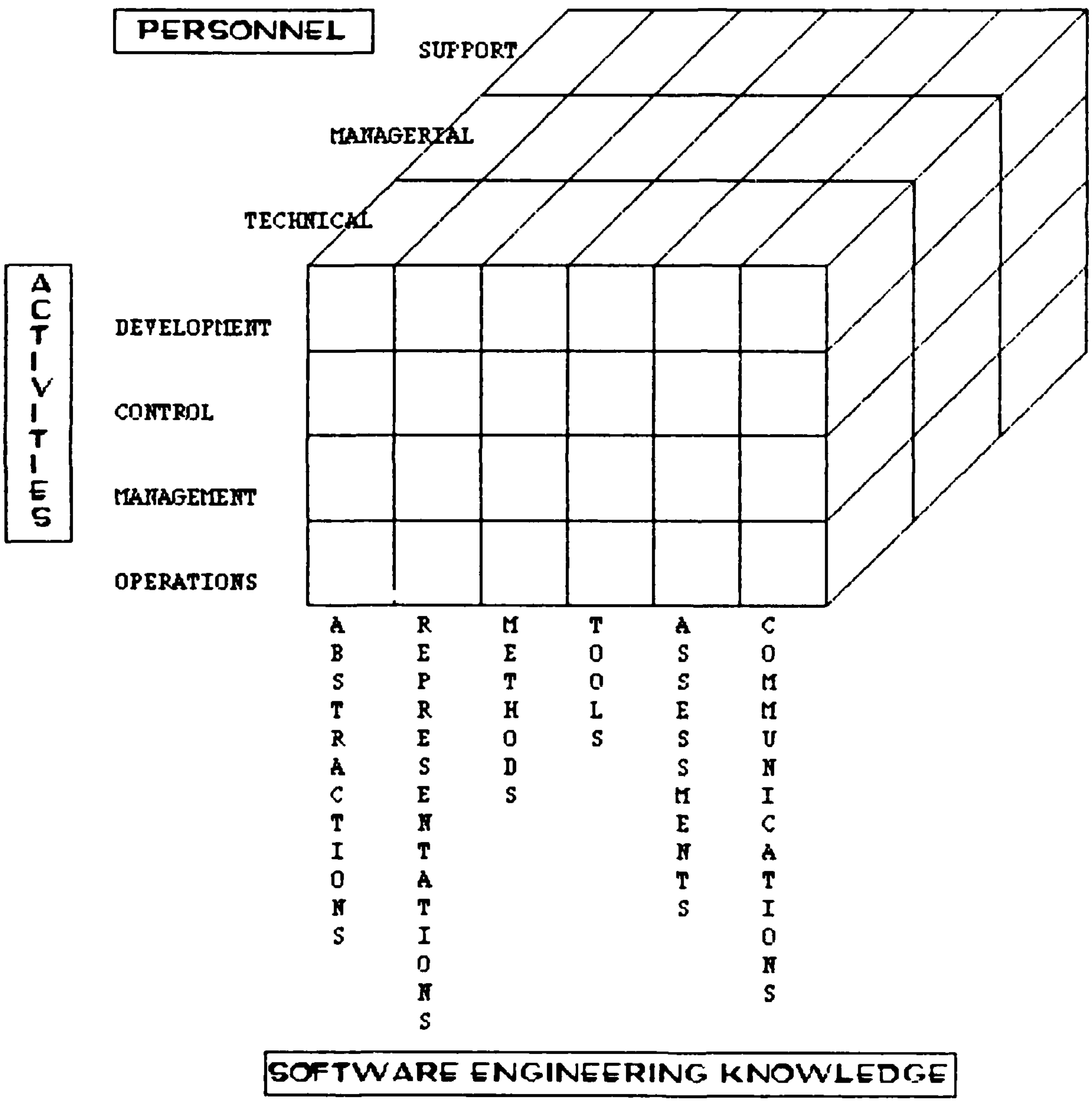
Managerial

- Life Cycle Overview and Implications
- Software Engineering Issues
- Methods and Tools
- The Role of Ada in Software Environment
- Managing Software Development
- Costing, Scheduling

Support

- Legal Issues
- Procurement Issues
- Technical Monitor
- Integration
- Safety and Security Issues
- User Services

SCHEMATIC OF CURRICULAR OPTIONS AVAILABLE FOR SOFTWARE ENGINEERING



Within each cell, three other dimensions are to be considered the environment (host, target, integration); the learner's entry skills (awareness, knowledge level, experience level, and mastery levels); complexity of the project (small, large, component, deputy).

PERSONNEL CATEGORIES

- TECHNICAL
- MANAGERIAL
- SUPPORT

SOFTWARE ENGINEERING ACTIVITIES

- Development
- Control
- Management
- Operations

Development

- Life Cycle Development Issues

Life Cycle Development Issues

- Requirements Analysis
- Specification Analysis
- Design
- Implementation
- Test

Control

- Reviews
- Quality Assurance
- V&V
- Reviews
- Safety and Security

Management

- Costing
- Scheduling
- Allocation

Operations

- Maintenance and Operations
- Training
- Installation
- Transition

SOFTWARE ENGINEERING KNOWLEDGE

- Abstactions - Principles and Models
- Representations - languages
- Methods
- Tools
- Assessment Models
- Communications Models

COMPLEXITY AND MAGNITUDE

- Projects in the Small
- Projects in the Large
- Program as Component
- Program as Deputy

LEVELS OF DEPTH

- Awareness
- Substantive Knowledge
- Relevant Experience
- Mastery and Integration

WHAT WILL A MODEL PROGRAM LOOK LIKE?

✓ A model education and training program will take at least a five year commitment to achieve.

✓ A program will take at least two years to demonstrate effectiveness.

✓ The commitment of upper-level management is essential.

✓ The commitment of resources are essential.

WHAT WILL A MODEL PROGRAM LOOK LIKE?

✓ A model education and training program will take at least a five year commitment to achieve.

✓ A program will take <u>at least</u> two years to demonstrate effectiveness.

✓ The commitment of upper-level management is essential.

✓ The commitment of resources are essential.

University Curriculum Support - Education

The role of the university is to provide a sound academic basis for students in software engineering. Students should graduate with the technical and communication competence necessary to achieve success in their chosen work settings.

University Curriculum Support - Education

- Bachelor's Program
- Master's Program - Modified SEI Model
- Conferences/Workshops/Seminars

Bachelor's Program

- Software Engineering Intro.
- Computer Science Intro.
- Programming Courses
- Programming-in-the-Large
- Design
- Data Structures
- Knowledge Representation
- SE Projects
- Communications

Master's Program - Modified SEI Model

This is a modular listing of topics. Though not meant to be offered as traditional university courses, the modules should be developed and offered within the confines of regular courses, as specified by the faculty.

Master's Program - Modified SEI... (cont'd)

- Project Economics
- Host-Target-Integration
- Requirements Analysis
- Specification
- System Design
- SW Design
- SW Implementation
- SW Testing
- System Integration
- Embedded Real-Time Systems
- Human Interfaces

Conferences/Workshops/Seminars

✓ The university should attract the best minds to discuss research and to address issues is an academic environment.

✓ Beyond courses, the university should promote conferences, workshops, meetings, seminars, and other academic gatherings in support of the goals of a comprehensive software engineering community effort.

✓ NASA should continue to encourage focused workshops and support conference attendance by employees as a means to infuse new ideas into the system.

NASA Curriculum Support - Training

- **Support for SE Training**
- Target Environment
- SSE with Procedures and Guidelines
- CAIS/APSE
- Ada as a PDL
- Safety and Fault Tolerance
- Operations and Maintenance
- Referencing and Library Support

Contractor Curriculum Support

- Internal SE Training Programs
- NASA Technical Monitors Invited
- Host Environments
- Integration Environments
- Data Structures

Community-Wide Support

- User Services
- Library Services
- Module Review
- Information Exchanges (BBS, newletters, CLSEATF)

KEY POINTS

A COMPLETE SOFTWARE ENGINEERING

TRAINING PROGRAM:

1. REQUIRES A SUBSTANTIAL COMMITMENT OF RESOURCES.

2. WILL TAKE THREE - FIVE YEARS TO TAKE EFFECT.

3. IS ESSENTIAL FOR THE SPACE STATION PROJECT.

4. RELIES ON TEAMS OF PEOPLE, NOT INDIVIDUALS

MSD

NASA MISSION SUPPORT DIRECTORATE JSC

THE SPACE STATION PROGRAM
SOFTWARE SUPPORT ENVIRONMENT (SSE)

RELATED EDUCATION AND TRAINING ISSUES

S.A. GORMAN
JUNE 10, 1987

NASA
National Aeronautics and
Space Administration

Lyndon B. Johnson Space Center
Houston, Texas

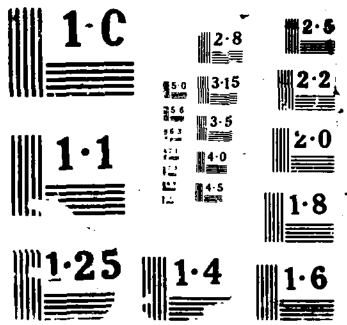
MISSION SUPPORT
NASA
JSC

OUTLINE

- SSE BACKGROUND
- SSE CONTRACT
- EDUCATIONAL ISSUES
- TRAINING ISSUES
- COURSES I'D LIKE TO SEE

AD-A189 658 ANNUAL ASEET (ADA SOFTWARE ENGINEERING EDUCATION AND TRAINING) SYMPOSIUM (SLIDES) HELD IN DALLAS TX ON 9-11 JUNE 1987(U) ADA JOINT PROGRAM OFFICE ARLINGTON VA 3/4

UNCLASSIFIED 11 JUN 87 F/G 12/5 NL


1·0
1·1
1·25
1·4
1·6
1·8
2·0
2·2
2·5
2·8
3·15
3·5
4·0
4·5

L81

BACKGROUND

- MARSHALL CONFERENCE ON SPACE STATION SOFTWARE ISSUES
  APRIL 1985
  - SPONSORED BY THE NASA SOFTWARE WORKING GROUP - SWWG
    - SOFTWARE MANAGEMENT
    - SOFTWARE DEVELOPMENT ENVIRONMENTS
    - LANGUAGES
    - SOFTWARE STANDARDS
  - PRINCIPAL RECOMMENDATIONS (PARAPHRASED)
    - NASA SHOULD FURNISH A UNIFORM, MODULAR SOFTWARE SUPPORT ENVIRONMENT AND REQUIRE ITS USE FOR ALL SPACE STATION SOFTWARE ACQUIRED OR DEVELOPED.
    - THE LANGUAGE ADA SHOULD BE SELECTED NOW (APRIL 1985) AS THE PRIMARY SOURCE LANGUAGE FOR SPACE STATION SOFTWARE...

CONTRACT MILESTONES

- SOURCE EVALUATION BOARD FORMED JANUARY 1986
- RFP RELEASED ON SEPTEMBER 11, 1986
- RESPONSES WERE RECEIVED ON NOVEMBER 12, 1986 - TWO RESPONDERS
    - TEAM LED BY IBM
    - TEAM LED BY LOCKHEED MISSILES & SPACE
- ORALS COMPLETED AND INTERIM SSE SYSTEMS DEMONSTRATED, FEBRUARY 1987
- BEST AND FINAL OFFERS RECEIVED APRIL 1987
- ESTIMATED CONTRACTOR SELECTION BY MID JULY 1987
- ESTIMATED CONTRACT START DATE BY MID AUGUST 1987

CONTRACT FORMAT

- CONTRACT IS SEPARATED INTO THREE PHASES
- PHASE 1 - REQUIREMENTS DEFINITION AND PRELIMINARY DESIGN
  - FIRST 15 MONTHS, COMPLETION FORM
- PHASE 2 - DESIGN, DEVELOPMENT, TEST AND EVALUATION (DDT&E)
  - STARTS ONE YEAR AFTER CSD, TERM SIX YEARS, LEVEL OF EFFORT
- PHASE 3 - SUSTAINING ENGINEERING
  - EXTENSION OF PHASE TWO
  - THREE YEARS - LEVEL OF EFFORT
  - OPTION OF THE GOVERNMENT

190

ADA IN THE SSE RFP

- REQUIREMENTS FOR THE USE OF ADA APPEAR IN SEC. 6.1.7, SSE SYSTEM CONCEPTUAL REQUIREMENTS
- ADA IS THE PRIMARY SOFTWARE LANGUAGE OF THE SPACE STATION PROGRAM
- THE FULL EXTENT OF ADA AND ITS ASSOCIATED SOFTWARE ENGINEERING PRINCIPLES SHALL BE REQUIRED
- OTHER LANGUAGE SUPPORT
  - EXISTING SOFTWARE
  - SPECIFIC APPLICATIONS
- ALL COMPILERS MUST BE VALIDATED
  - HOSTS AND TARGETS TBD
  - OPTIONAL FEATURES TBD

EDUCATIONAL ISSUES

- NEW AND UNTRIED WAY FOR NASA TO DO SOFTWARE
  - WILL REQUIRE AN EXTRAORDINARY LEVEL OF INTERCENTER COOPERATION
  - METHODS LOCALIZED TO SPECIFIC CENTERS - HIGHLY INDIVIDUAL
    - HAL/S AT JSC
    - GOAL AT KSC
    - STOL AT GSFC
- COMMONALITY RECOGNIZED AS NOBLE GOAL WITHIN NASA
  - HAS NOT BEGUN TO HURT YET
  - SMALL SYSTEM MIND-SET
  - ACCEPTANCE OF "VALUE-ADDED" CONCEPT

MORE EDUCATIONAL ISSUES

- LIVING WITH THE VON-NEUMANN COMPUTER FOR A FEW MORE YEARS
- THE STATION ENVIRONMENT WILL BE A WIDE AREA NETWORK OF LOCAL AREA NETWORKS
  - MANY ISSUES - NOT NEAR ENOUGH SMART PEOPLE
  - UNPRECEDENTED SYSTEM VULNERABILITY - UP THERE ALL THE TIME
  - LEVEL B3 MULTI-LEVEL SECURITY?
  - DISTRIBUTED DATABASES
- EXCITING NEW DOCUMENTATION TECHNOLOGIES
  - OPTICAL STORAGE
  - MIXED TEXT & GRAPHICS, "DESKTOP PUBLISHING TECHNIQUES"
  - HYPERTEXT APPROACHES
  - VIRTUALLY UNKNOWN IN THE AGENCY

TRAINING ISSUES - SKILLS ENHANCEMENT

- BETTER SHAPE HERE - MANY COURSES AVAILABLE
  - MARKET WILL DRIVE THIS PROBLEM
- WORK STATION WORLD - MANAGERS AS WELL AS TECHNICAL PERSONNEL
  - REQUIRED FOR SUCCESS OF SSE
  - RELATED SKILLS REQUIRED - TYPING, PROFS, TELEMAIL, LOTUS ETC.

Tie into Roles

- ADA MORE COMPLEX - CERTIFICATION?
  - ESTABLISHED TRAINING SEQUENCES - TESTING MILESTONES
- SOFTWARE REUSE
  - DESIGNED INTO THE PRODUCT FROM THE START
  - UNDERSTANDING THE HIDDEN COSTS
    - NASA CENTER TO NASA CENTER
    - DOD, AEROSPACE INDUSTRY, NATO ETC.

.93

COURSES I'D LIKE TO SEE

- REALISTIC APPROACH TO CONTRACT COSTING
- UNDERSTANDING LARGE SOFTWARE PROJECTS
- COMPUTER AIDED TRAINING - "HOW TO"
- EXPERT SYSTEMS AS "VALUE ADDED" COMPONENTS

REALISTIC APPROACH TO CONTRACT COSTING

- INSPIRED BY QUOTE FROM MAY, 1987 ISSUE OF "DEFENSE SCIENCE & ELECTRONICS", BY DR. Y.J. LUBKIN:

  "THE MAIN CONCLUSION, WHICH EVERYBODY INVOLVED IN PROPOSAL EFFORT ALREADY KNOWS, IS THAT IN ORDER TO GET THE JOB YOU HAVE TO LIE. IF YOU ARE HONEST, YOU ARE ALSO BROKE.

  ANOTHER EX-BOSS'S BID PHILOSOPHY WAS TO FIND OUT HOW MUCH MONEY THE CUSTOMER HAS, BID THAT AMOUNT, PROMISE HIM ANYTHING, AND MAKE UP THE DIFFERENCE ON ENGINEERING CHANGE ORDERS. IT STILL SEEMS TO WORK."

- IT REALLY DOES WORK THIS WAY

- NEED BETTER LIFE CYCLE SOFTWARE PROJECT COSTING MODELS AND PEOPLE IN DECISION MAKING ROLES WHO UNDERSTAND THEM

- UNDERSTAND THE EFFECTS OF THIS TYPE BUDGETING ON THE PROJECT OVER ITS LIFETIME

UNDERSTANDING LARGE SOFTWARE PROJECTS

- FOR SMALL PROJECT PEOPLE!
  - MORE THAN YOU THINK
- LARGE PROJECT REALLY IS DIFFERENT
  - SHUTTLE ORBITER HAS 65 IBM SUSTAINING ENGINEERS TODAY
    - APPX 450K LINES OF CODE
    - TEST ENVIRONMENT TOTALLY DIFFERENT
- MAKE THE DIFFERENCES COME ALIVE
- PROVIDE AN APPRECIATION OF WHAT ADA AND PROJECTS LIKE THE SSE ARE ATTEMPTING TO ACCOMPLISH

COMPUTER AIDED TRAINING - "HOW TO"

- SURVEY THE DIFFERENT TYPES OF COMPUTER AIDED TRAINING
- UNDERSTANDING OF RELATIVE COSTS
- TRAINING COMBINATIONS - PARTICULARLY WITH VIDEO
- COURSE DEVELOPMENT
  - REQUIREMENTS
  - DESIGN
  - IMPLEMENTATION
  - MAINTENANCE

858

EXPERT SYSTEMS AS "VALUE ADDED" COMPONENTS

- EXPERT SYSTEM CLASSIFICATION
  - WHICH ONES REQUIRE SYMBOLIC PROCESSORS
  - RESOURCE UTILIZATION
  - LANGUAGE DEPENDENCIES
- VERIFICATION AND VALIDATION
  - SELF MODIFYING PROGRAMS
  - CONSTRAINTS OF TESTABILITY - WHAT'S LEFT?
- IMPLEMENTATION IN ADA - MAINTAINED IN STANDARD ENVIRONMENT

IMPLEMENTING A LIFE CYCLE MODEL FOR SOFTWARE ENGINEERING AND Ada* TRAINING: AN OVERVIEW OF THE ISSUES

by

DR. CHARLES W. McKAY

SOFTWARE ENGINEERING RESEARCH CENTER

(SERC)

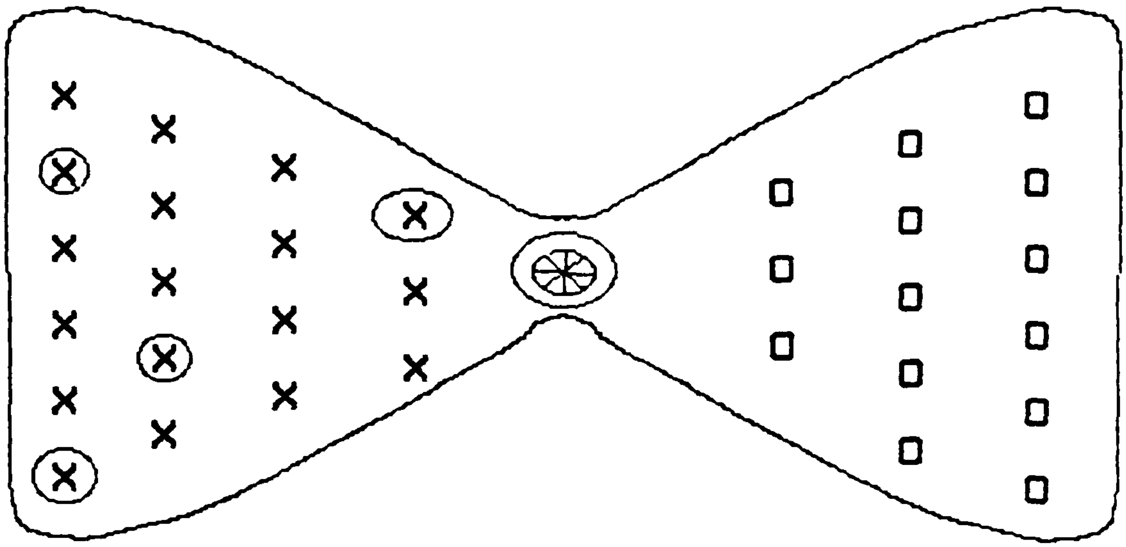
UNIVERSITY OF HOUSTON - CLEAR LAKE

*Ada is a registered trademark, U.S. Government, AJPO

TWO SCENARIOS FOR

SSP ENVIRONMENT

IN 2000+ A.D.



HOST ENVIRONMENTS:

- DEVELOP
- SUSTAIN

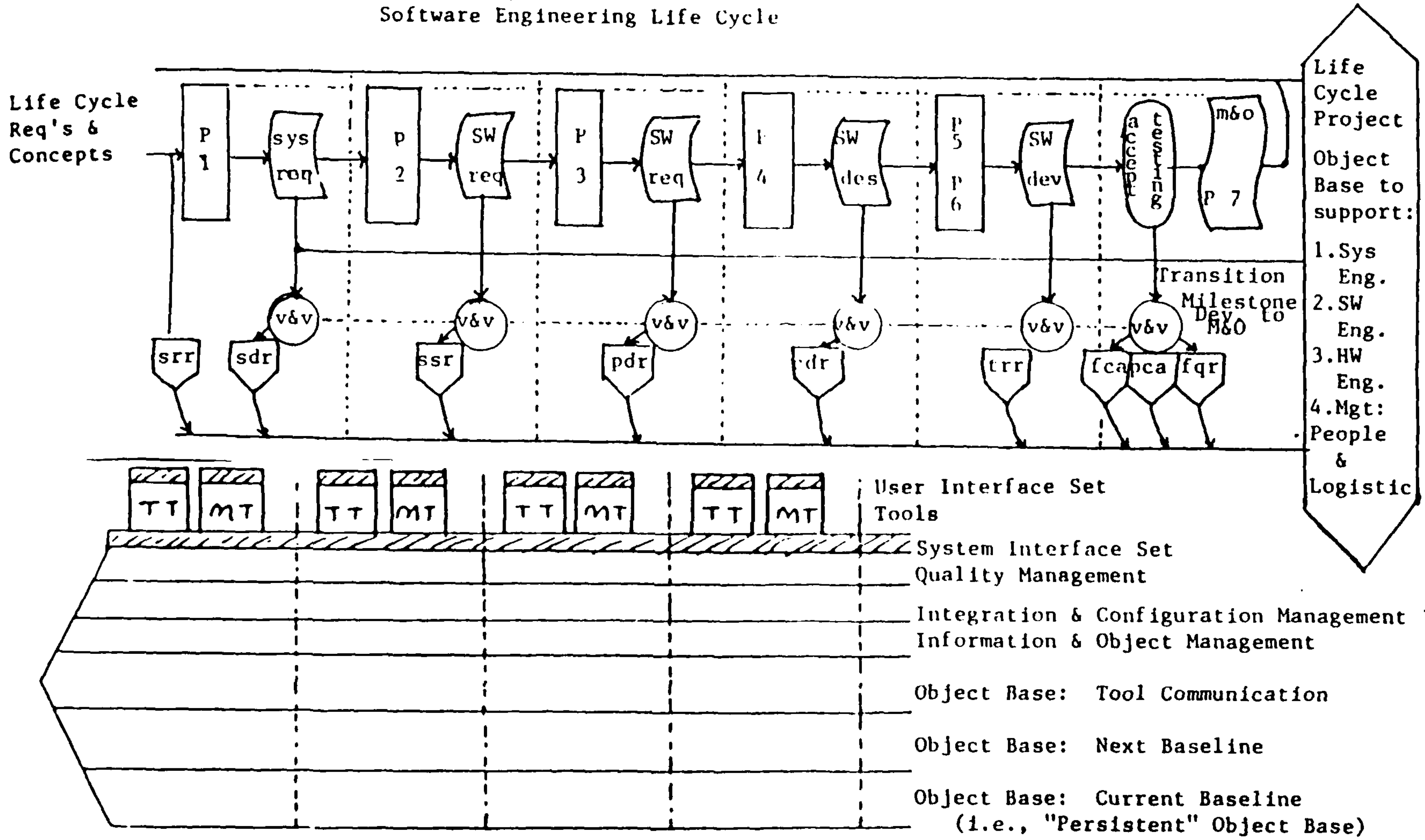
INTEGRATION ENVIRONMENT:

- CONTROL OF TGT. ENVIR. BASELINE
- INTEGRATION V&V FOR NEXT BASELINE AND TEST & INTEGRATION PLANS

TARGET ENVIRONMENTS:

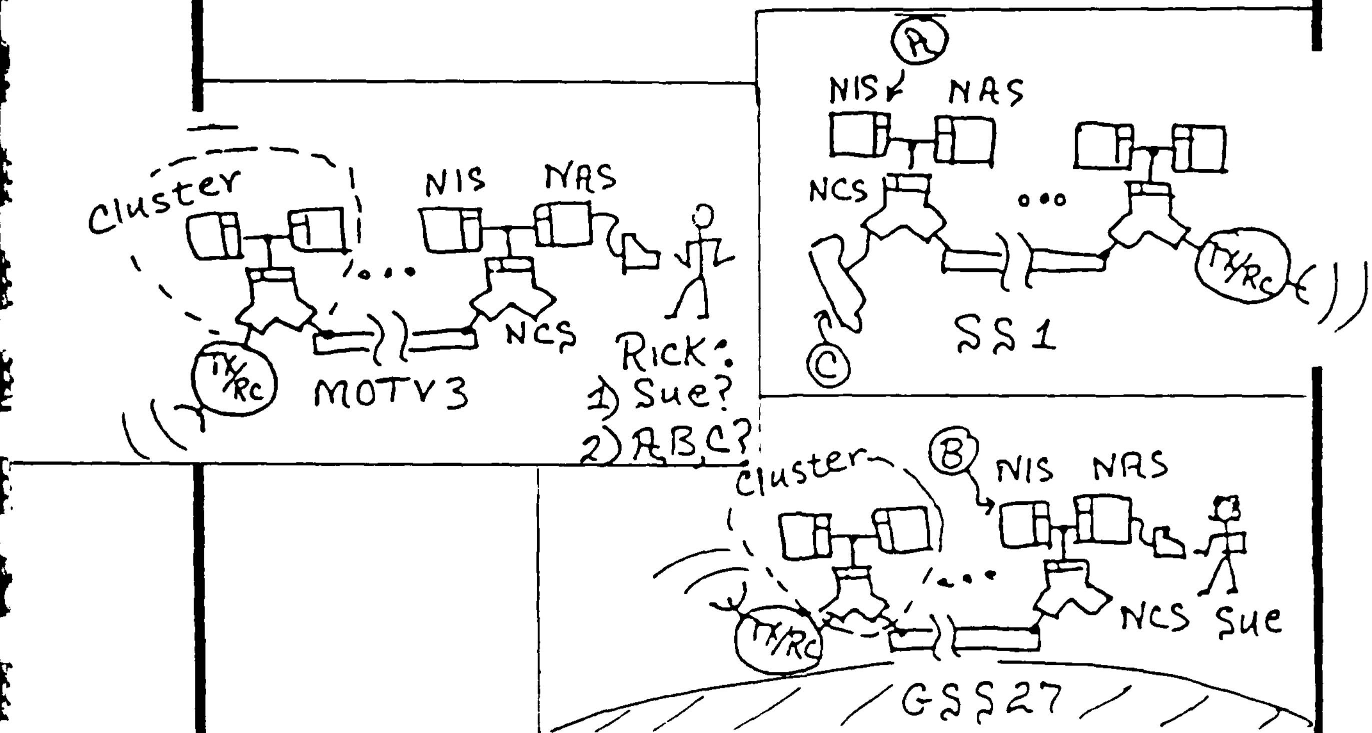
- DEPLOY
- OPERATE

A Conceptual Model of the
Software Engineering Life Cycle



An Implementation Model
Of an SSE Based Upon the
Above Conceptual Model

PROPOSED SCENARIOS FOR SPACE STATION SYSTEMS



- 12 TYPES LAN'S
- SEVERAL LAN INSTANTIATIONS
- 23 CLUSTER: SS
- 10 YEAR DEVELOPMENT SS, 15 YEAR LIFE CYCLE, ∞ LIFE CYCLE
- TRANSPARENCY (at user friendly interface level) OF:
  - ✓LOCATION ✓PARALLELISM
  - ✓REPLICATION
  - ✓ FAULT TOLERANCE

SLIDES

TRACK I - INDUSTRY

LESSONS LEARNED

THURSDAY, JUNE 11, 1987

ADA TRAINING:
A DEVELOPMENT TEAM'S PERSPECTIVE

Presented By:
Rudy Vernik
CCSO / SKAS
TINKER AFB OK 73145
PH (405) 734-2457

223

OVERVIEW

- BACKGROUND
- ADA TRAINING NEEDS
- TRAINING EXPERIENCES
- RECOMMENDATIONS
- QUESTIONS


SARAH

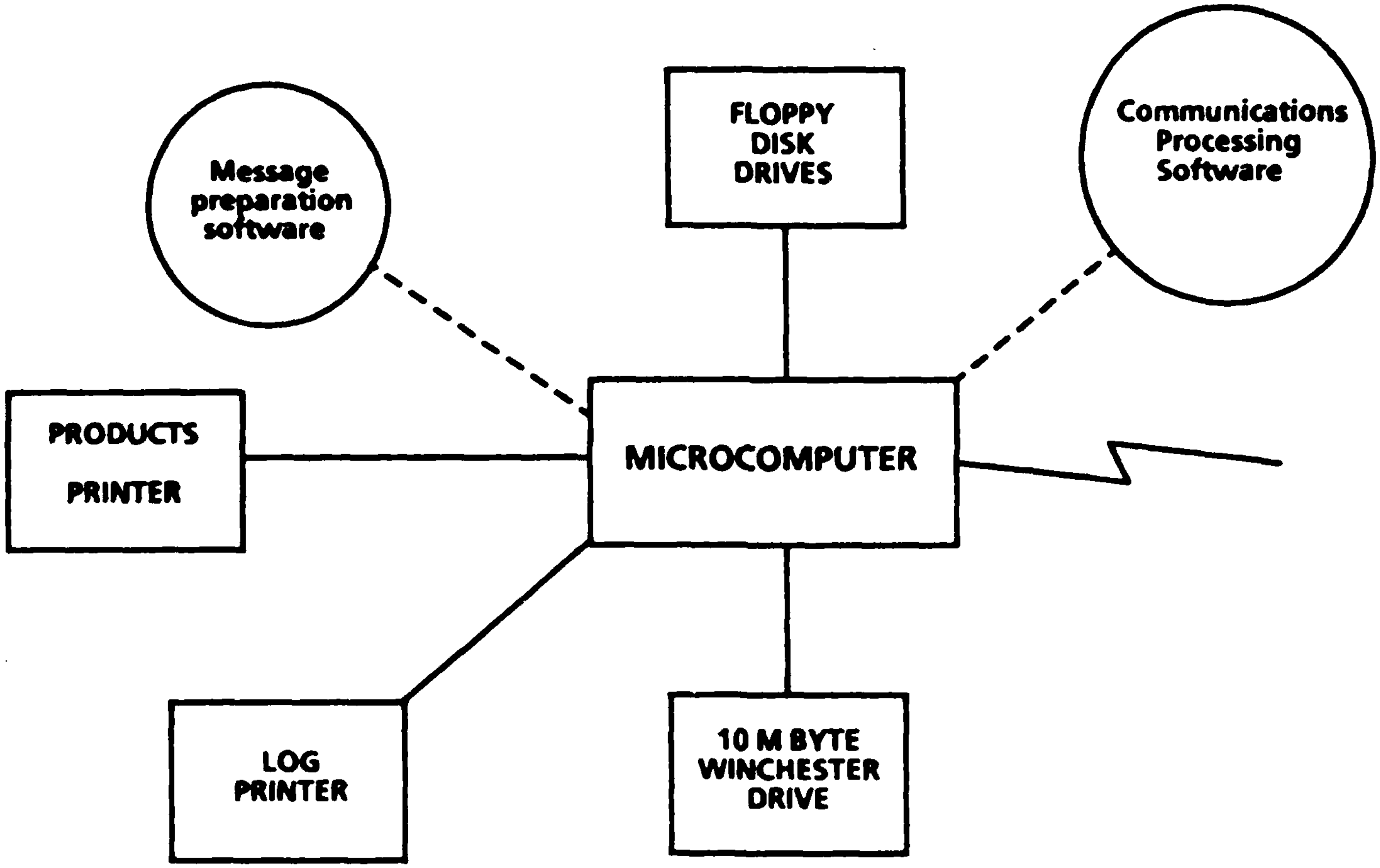
TWO VERSIONS OF SARAH

- COMMUNICATIONS VERSION
  - FULL DUPLEX COMMUNICATIONS
    - SOFTWARE MODE 1
  - DISTRIBUTION FUNCTION
  - MESSAGE PREPARATION / EDITING
  - UTILITES
- ADMINISTRATION VERSION
  - MESSAGE PREPARATION / EDITING
  - UTILITES

R6

SARAH COMMUNICATIONS WORKSTATION

(BASIC WORKSTATION)



ADA TRAINING NEEDS

- SOFTWARE ENGINEERING
- MANAGEMENT TRAINING
- DEVELOPMENT METHODS
- TOOLS TRAINING
- LANGUAGE TRAINING

SOFTWARE ENGINEERING

- UNDERGRADUATE AND POSTGRADUATE COURSES
- CONTENT - FUNDAMENTALS
- ACCREDITATION
- GOVERNMENT AND INDUSTRY SUPPORT

MANAGEMENT TRAINING

- SCHEDULING
- USE OF PROTOTYPES
- WAYS OF DETERMINING PROGRESS
- POSSIBLE PROBLEMS
- DETERMINING EQUIPMENT NEEDS

HOW BEST TO SUPPORT THE ADA DEVELOPMENT TEAM

DEVELOPMENT METHODS

- WHICH METHODOLOGY?
  - OOD
  - JSD
  - PAMELA
- COVER SEVERAL METHODS
- USE OF PDLS

TOOLS TRAINING

"HOW MUCH TRAINING DO YOU NEED TO USE A COMPILER ?"

- CONFIGURATION MANAGEMENT
- DOCUMENTATION CREATION
- DEBUGGERS
- ANALYSIS TOOLS

LANGUAGE TRAINING

- BASE ON FUNDAMENTAL PRINCIPLES AND PRACTICES
- BASIC AND ADVANCED COURSES
- 80 HOURS MINIMUN
  - WELL ORGANIZED
  - ALLOW FOR CONSOLIDATION
  - MOTIVATED STUDENTS

TRAINING EXPERIENCES
FORMAL TRAINING

- SELECTION
  - CREASE
  - SIGADA
- TRAINING SPECIFICATION
- COSTS
- STUDENT EVALUATION
- PROCUREMENT DELAYS

TRAINING EXPERIENCES
IN-HOUSE TRAINING

- INFORMAL LECTURES
- COMPUTER AIDED INSTRUCTION
- VIDEO TAPES
- SELF STUDY
- CONFERENCES

RECOMMENDATIONS

- ALLOW SUFFICIENT FUNDS FOR ALL TRAINING NEEDS
- BASE ADA TRAINING ON SOUND SOFTWARE ENGINEERING PRINCIPLES
- PROVIDE TRAINING FOR MANAGERS
- CAREFULLY EXAMINE TRAINING ORGANIZATIONS FOR EXPERIENCE AND APPROACH
- PROVIDE CAI PACKAGES FOR CONSOLIDATION
- ALLOW CONSOLIDATION TIME
- FOCUS ON THE TASK, NOT THE TOOL

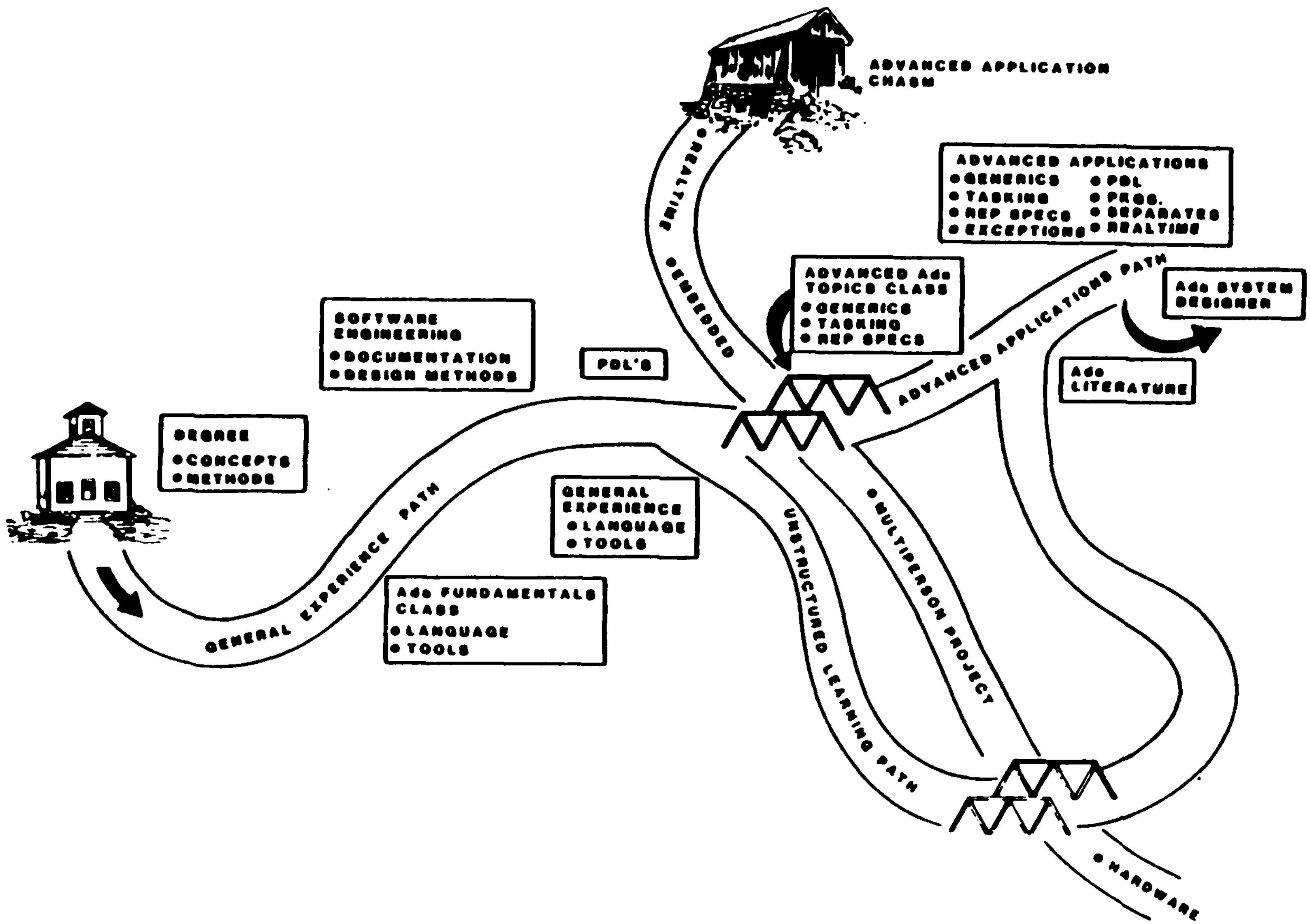
Ada* for the Manager

A Texas Instruments Perspective

Freeman L. Moore
Texas Instruments Incorporated
Plano, Texas 75086

* Ada is a registered trademark of the U.S. Government, AJPO

Ada Training Roadmap



Defense Systems & Electronics Group

FLM - 06/11/87

The excuses:

- o I don't have the time
- o I don't need this course
- o only portions seems appropriate for me
- o I can read, so why do I need this class
- o Ada won't impact my work
- o Ada is just a language

The Problem:

Software Engineer vs Software Manager

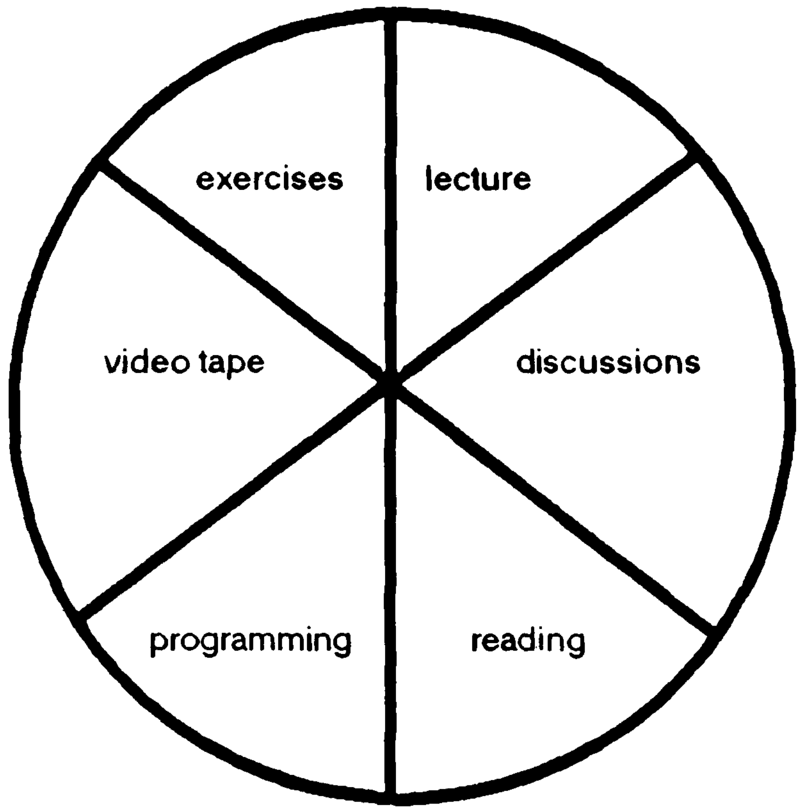
Programming skills	Analysis and design
Design skills	Management skills
testing skills	language awareness

The Solution:

Software Engineer vs Software Manager

Software Engineer	Software Manager
Software Engineering with Ada	Ada Manager's Overview
Advanced Topics in Ada	Technical Proposals Workshop
Design with Ada (PDL)	
Embedded systems	

Instructional Design topics



Generics

Tasking features

Representation specs

case study design

Advanced

Fundamentals

Software Engineering Principles

language skills

tool usage

Object – oriented design

- HS/SW simulation tools
- Specifics of compiler features
- Optimizing an implementation
- Underlying hardware architecture

Design with Ada

- Analysis and design methods
- Using PDL constructs
- Working with PDL tools
- Interfacing design techniques

Embedded Systems

Manager's Overview

- Terminology
- Language capabilities
- Relevant TI and DoD standards
- Other Projects experience

Other resources

- Proposal Issues Workshop
- Internal Newsletter
- Central Ada Strategy contact

Manager's Overview

Minimal history

Understanding the role of PDL

Background on various standards and directives

References to DoD Ada – related contracts

Language capabilities

Terminology

Design techniques

Helping to Understand Ada's impact

- o internal newsletter
- o trip reports from people attending conferences
- o customized training
- o awareness of training possibilities
- o involvement with SEI
- o working with APSE tools

LESSONS LEARNED

- Training must be justified
  - time away from job

- Training needs change
  - incorporate project experience
  - background of audience changes

- Training must fit the need of the audience
  - must be relevant
  - must provide timely information
  - must reflect the work environment

Ada in the MIS World

Eugen N. Vasilescu

Ada Lab
Grumman Data Systems

GNV Associates

Hofstra University

M I S

COBOL

Ada DBMS

4GL

Ada and COBOL

Pros

Ada

+ Strong Typing
+ Packages (Data Abstraction)
+ Separate Compilation
+ Concurrent Processing
+ Exception Handling
+ Portability
+ Efficient Space+Time

2

Ada and COBOL

Pros

COBOL

+ Entrenched
+ Large Pool of Prof
+ Good File Handling
+ Well Integrated with DBMS

Ada and COBOL

Cons

Ada

- Lack of Ada experience in the MIS world

- Lack of Education Materials

- Lack of Standard Indexed I/O Packages

Ada and COBOL

Cons

COBOL

- Maintenance Problems
- Lack of Portability
- Limited Capabilities

Some Current Ada Indexed I/O

K. Kurbel and W. Pietsch

+ Uses B-Trees

+ Compatible with DIRECT_IO

+ Good use of Generics and Private Types.

- No support for multiple keys

- No Dynamic Key Manipulation

Some Current Ada Indexed I/O

A. Keller, G. Wiederhold et al.

\+ Multiple Indexes

\+ Dynamic Indexes

\+ Concurrency Support

\+ Variant Record Support

\- Unimplemented

\- Not "for" Ada

4GL Features

No accepted definition

Non Procedural

User Friendly

Used together with
a DBMS

Integrates severa
functions

Fast Code Development

Domain Specific

4GL Cons

No Strong Typing

Not reliable

Not portable

Wasteful of time and space

Weak Information Hiding

Ada and 4GLs Complement each other

DBMS Approaches

- # Hierarchical (IMS)
- # Network (IDMS)
- # Relational (DB2, Oracle, INGRES) SQL Standard
- # Entity-Relationship
- # Semantic Model
- # Knowledge-Base DBMS

Ada compatible DBMS Directions

P.Buneman, M.Atkinson

Need to Combine Relational DBMS and Object-Oriented Programming

Need for Type Checking to help insure program quality

Need For Data to Survive Programs (Packages as Systems)

Ada compatible DBMS Directions

P.Dadam et al

Limitations of the Relational approach

Need to represent Hierarchical structures

Give up the first normal form to allow relations as attributes

Extend SQL to handle the new kind of relation

Ada/DBMS Directions

IDA Ada/SQL Prototype

Use Ada Strong typing

Relations are Ada records: support for hierarchical models

Use Ada separate compilation and packages

Pure Ada following a SQL-like syntax.

Easy interfaces to COTS DBMS

Ada/DBMS Directions

WIS Ada DBMS ?

Use Ada Strong typing

Support of all DBMS models

Use Ada separate compilation and packages

Can be taylored to a variety of Hardware/ Software Environments

Use of a layered approach

Conclusions

Ada is a strong contender in the MIS world

Need for a standard Indexed I_O package

Need for immediate ties and interfaces to existing DBMS

Ada*
TRAINING
FOR THE
AFATDS PROJECT

* Ada is a registered trademark of the U.S. Government (AJPO).

presented by

DONALD G. FIRESMITH

Magnavox Electronic Systems Company
M/S 10-C-3 Dept. 566
1313 Production Road
Fort Wayne, IN 46808
(219) 429-4327

Ada Training for the AFATDS Project

- AFATDS Overview
- AFATDS Ada Training
- AFATDS Lessons Learned

AFATDS Overview

- The Advanced Field Artillery Tactical Data System (AFATDS) is a very large automated command and control system for the US Army.
- Contractor: Magnavox Electronic Systems Company
- Contact person: Mr. Skip Carstensen, AFATDS Software Director, (219) 429-5272
- AFATDS will serve as both a fire support control and coordination system (cannon, rocket, missile artillery, mortars, air support, and naval gunfire) and a field artillery command and control system.
- AFATDS will support all levels of command from platoon to corps.

- AFATDS is a multi-phase project nearing completion of Concept Evaluation Phase (CEP).
- AFATDS consists of seven major components that can be configured from a single component to a large center contained in several tactical vehicles.
- Host development system: DEC VAX / VMS
- Target system: Motorola MC68020-based (32 bit) workstation with touch-entry graphics display.
- The design objectives were to optimize operation efficiency, simplify training, ease maintenance, reduce life-cycle costs, and improve battlefield survivability.
- AFATDS was the largest new Ada development contract in terms of lines of code when let in 1984.
- The main software development method was Object-Oriented Development (OOD).

- Software Size (non comment, non blank):
  - 110 K SLOC at Release 1 (Feb 1986)
  - 253 K SLOC at Release 2 (Apr 1986)
  - 493 K SLOC at Release 3 (Aug 1986)
  - 770 K SLOC at Release 4 (Aug 1987)
- Productivity (per person-month):
  - 275 SLOC during Release 1
  - 800 SLOC during Release 2
  - 925 SLOC during Release 3

AFATDS Ada Training

1. The Magnavox Ada training program was started in 1980.
2. The AFATDS project used the Telesoft-Ada compiler for early training. Subsequent classes used the DEC Ada compiler.
3. The AFATDS Ada training program was primarily provided by EVB Software Engineering.
4. The initial classes were primarily attended by technical managers.
5. Magnavox has now developed it's own training capability and retains a number of academic consultants in Ada and software engineering from a local university.

ADAS = Analysis and Design for Ada Software (1 week)
ADW = Ada Programming Workshop (1 week)
AAPW = Advanced Ada Programming Workshop (1 week)
CDE = Custom Design Examples (1 week)
TEST = Ada Testing (1 day)

Qtr	Yr	ADAS	APW	AAPW	CDE	TEST
1	84	21	20	21		
2	84				21	
3	84	16	12			
4	84			12		
1	85	24	21	21		
2	85	44				
3	85		39	40		
4	85	20	18			
1	86			19		
2	86	23	22			87
Total		148	132	113	21	87

AFATDS Lessons Learned

1. Ensure management support.

   Management support for Ada is essential. Ada will not be used as intended and innovative software development methods will not be successfully implemented without active management support at all levels.

2. "Enlighten" corporate and project management regarding:

   (a) Ada Project Management

   (b) Benefits and Risks associated with Ada

   (c) Ada Application Areas

   (d) Software Engineering (Overview)

   (e) DoD Software Development Standards (Overview)

3. Ensure technical management competence.

Low and mid-level technical managers must remain technically current on projects involving a new software development method, a new language, and a new mindset. They must not become bogged down in micro-scheduling and status reporting.

4. Train technical management in:

(a) Ada Project Management

(b) Software Engineering

(c) Ada-Oriented Development Methodology

(d) Ada Mindset and Culture

(e) Ada Programming Support Environment

(f) Ada Language (Overview)

(g) DoD Software Development Standards

5. Ensure government and IV&V understanding.

The contractor's risk increases greatly if pertinent government and IV&V personnel do not receive training in software engineering, the project software development method (if state-of-the-art), and Ada.

6. Offer to train customer and IV&V personnel in:

(a) Software Engineering

(b) Ada-Oriented Development Methodology

(c) Ada Mindset and Culture

(d) Ada Language (Overview)

(e) DoD Software Development Standards (impact on Ada)

7. Train system designers in:

   (a) Software Engineering (Overview)

   (b) Ada-Oriented Development Methodology

   (c) Ada Mindset and Culture

   (d) Ada Language (Overview)

   (e) DoD Software Development Standards (Overview)

8. Train software designers, programmers, and testers in:

   (a) Software Engineering

   (b) Ada-Oriented Development Methodology

   (c) Ada Mindset and Culture

   (d) Ada Programming Support Environment

   (e) Ada Language

   (f) DoD Software Development Standards (Overview)

9. Train Software Quality Assurance and Software Configuration Management personnel in:

   (a) Software Engineering

   (b) Ada-Oriented Development Methodology

   (c) Ada Mindset and Culture

   (d) Ada Language

   (e) DoD Software Development Standards (Overview)

10. Train marketing personnel in:

   (a) Benefits and Risks associated with Ada

   (b) Ada Application Areas

   (c) DoD Software Development Standards (Overview)

11. Spend adequate time and money prior to project initiation to:

    (a) Determine the appropriate software development method.

    (b) Develop software standards and procedures (e.g., Ada coding standards).

    (c) Integrate the method, standards, and procedures into the training.

12. Hire or assign inhouse instructors and experts.

13. Use consultants to train inhouse instructors and experts.

14. Provide COMPLETE classroom training to developers.

15. Provide CONTINUING on-the-job training for developers.

16. Provide methodology and Ada help-desks.

17. Set-up regular inhouse seminars to spread lessons learned.

18. Provide instructors and developers ready access to methodology and Ada related journals and newsletters.

19. Send instructors, experts, and as many developers as practical to conferences, workshops, outside seminars, etc.

20. Provide for constant developer feedback into methodology and training courses.

21. Provide for constant instructor and expert oversight into developer implementation of methodology and Ada training.

22. The cost of proper training is usually underestimated.

23. The amount of training required is usually underestimated.

24. The scope of training required is usually underestimated.

25. Developer proficiency is usually overestimated (at the beginning).

26. Developers do not need to master the entire LRM to be effective.

27. Recent graduates become effective Ada practitioners faster because of the widespread emergence of Pascal and Software Engineering curricula in the 1980's. Several AFATDS developers had recently studied both Ada and OOD at a local university.

28. The most common cause of trainee failure is long experience with one language (especially assembly) coupled with an inflexible attitude.

29. The second most common cause of trainee failure is negative management attitudes towards Ada and modern software engineering.

30. Select textbooks that:

(a) Emphasize software engineering first; it is far more important than Ada syntax and semantics.

(b) Develop an Ada mindset.

(c) Teach one or more modern Ada-oriented development methods.

(d) Explain why the language is the way it is.

(e) Teach the complete language (e.g., tasking, generics, etc.).

(f) Refer to the Language Reference Manual (LRM).

(g) Teach proper Ada style.

(h) Contain numerous practical examples from the proper application area(s).

(i) Contain only examples that have been compiled on a validated compiler.

(j) Have been written by an author with practical experience in the appropriate application areas.

(k) Are current (The field is rapidly progressing).

(l) Emphasize designing and coding for reusability and portability.

31. Avoid textbooks that:

    (a) Only teach syntax.

    (b) Only include software engineering as an afterthought.

    (c) Teach AdaTRAN, PasAda, etc.

32. Recommended textbooks include:

    (a) "Ada Rationale" by Jean Ishbiah

    (b) "Software Engineering with Ada" by Grady Booch

    (c) "Object-Oriented Development Handbook" by Ed Berard

    (d) "Programming in Ada" by J.G.P. Barnes

33. Recommended references include:

    (a) Ada Language Reference Manual (LRM)

    (b) "Ada as a Second Language" by Norman H. Cohen

34. Proper training pays.

35. Keep the expertise you develop.

SLIDES

TRACK II - ACADEMIA

LESSONS LEARNED

THURSDAY, JUNE 11, 1987

Lessons Learned Panel:

Academic Track

- Dr Charles McKay, Chair
- Major Charles Engle
  & Major Colen Willis
- Dr Robert Mers
- Dr Charles Kirkpatrick
  & Dr Paul Knese
- Mr Victor Meyer
- Mr David Barrett
- Dr Clifford Layton

Second Annual ASEET Symposium

9..11 June 1987

Presentations:

Dr Charles McKay, UH Clear Lake, SERC (Software Engineering Research Center)
"A Perspective of the Role of Ada* in a Software Engineering Curriculum"

Major Charles Engle & Major Colen Willis, US Military Academy, West Point
"Turning COBOL Programmers into Ada* Software Engineers"

Dr Robert Mers, Dept. of C.S., North Carolina A&T State University
"Teaching Software Engineering in a First Ada* Course"

Dr Charles Kirkpatrick & Dr Paul Knese, Parks College, Saint Louis University
"Ada* Education & the Non-Computer Scientist"

Mr Victor Meyer, Saint Mary College
"Ada* in Undergraduate Curriculum at Saint Mary College"

Presentations Cont:

Mr David L Barrett, Dept. of C.S., East Texas State University
"The Programming Team & the Accelerated Course as Methods for Teaching Ada* "

Dr Clifford Layton, Rogers State College
"Teaching Ada* in the University"

Question & Answer Session

A Personal Perspective of the Role of Ada* in a Software Engineering Curriculum

by

Charles W McKay, SERC

Outline:

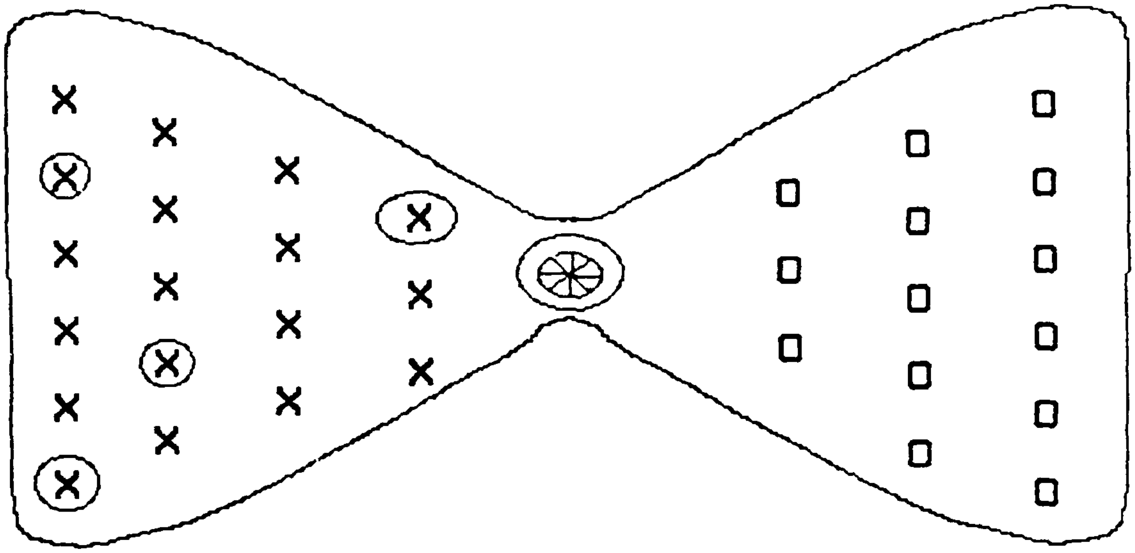
- Some Observations about the Relation of Software Engineering to: Computer Science, Computer Sys. Eng.

- Context of a Software Eng. Curriculum: Life Cycle Concerns of 3 Environ.

- Approaches to Teaching Ada as a Lang: Syntax .. Rationale, Pgm-in-Small .. Team Projects

- Approaches to Building on Ada Culture as Cornerstone of a Sw. Eng. Curric: Whole-Part-Whole ⇒ Top Down Iterations of Breadth, Bottom Up Implementations of Depth

TWO SCENARIOS FOR SSP ENVIRONMENT IN 2000+ A.D.



HOST ENVIRONMENTS:

- DEVELOP
- SUSTAIN

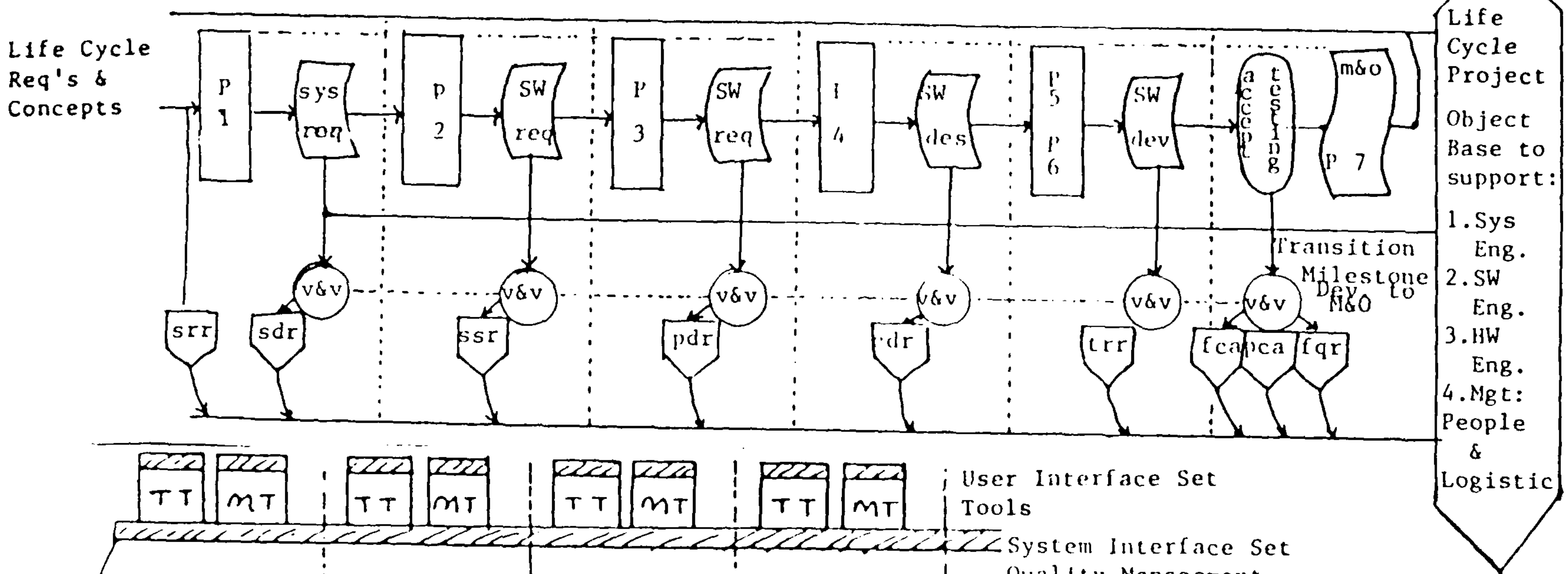
INTEGRATION ENVIRONMENT:

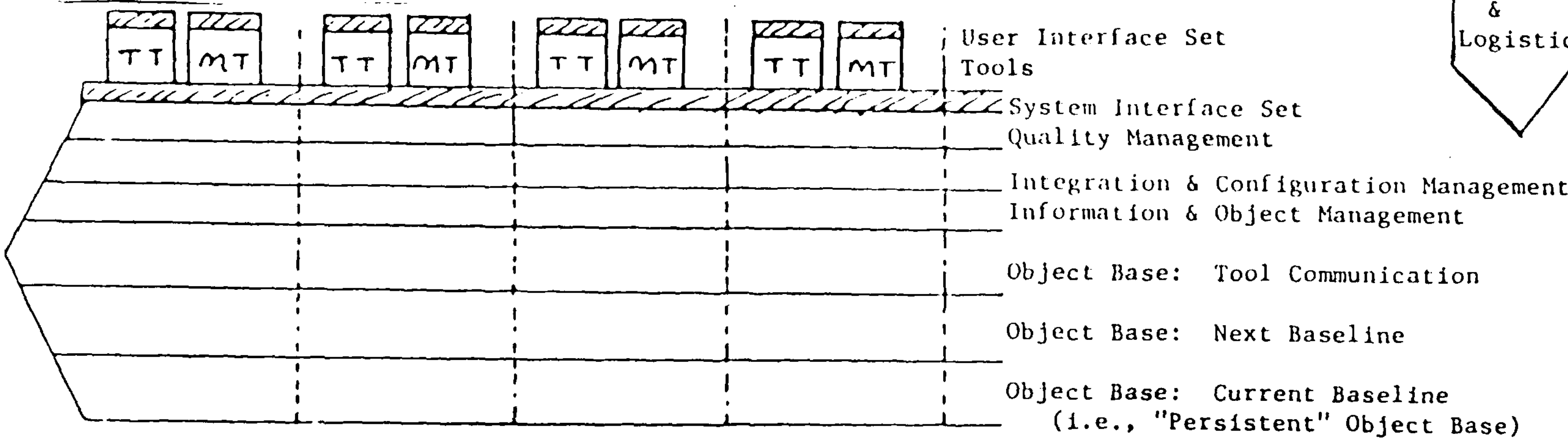
- CONTROL OF TGT. ENVIR. BASELINE
- INTEGRATION V&V FOR NEXT BASELINE AND TEST & INTEGRATION PLANS

TARGET ENVIRONMENTS:

- DEPLOY
- OPERATE

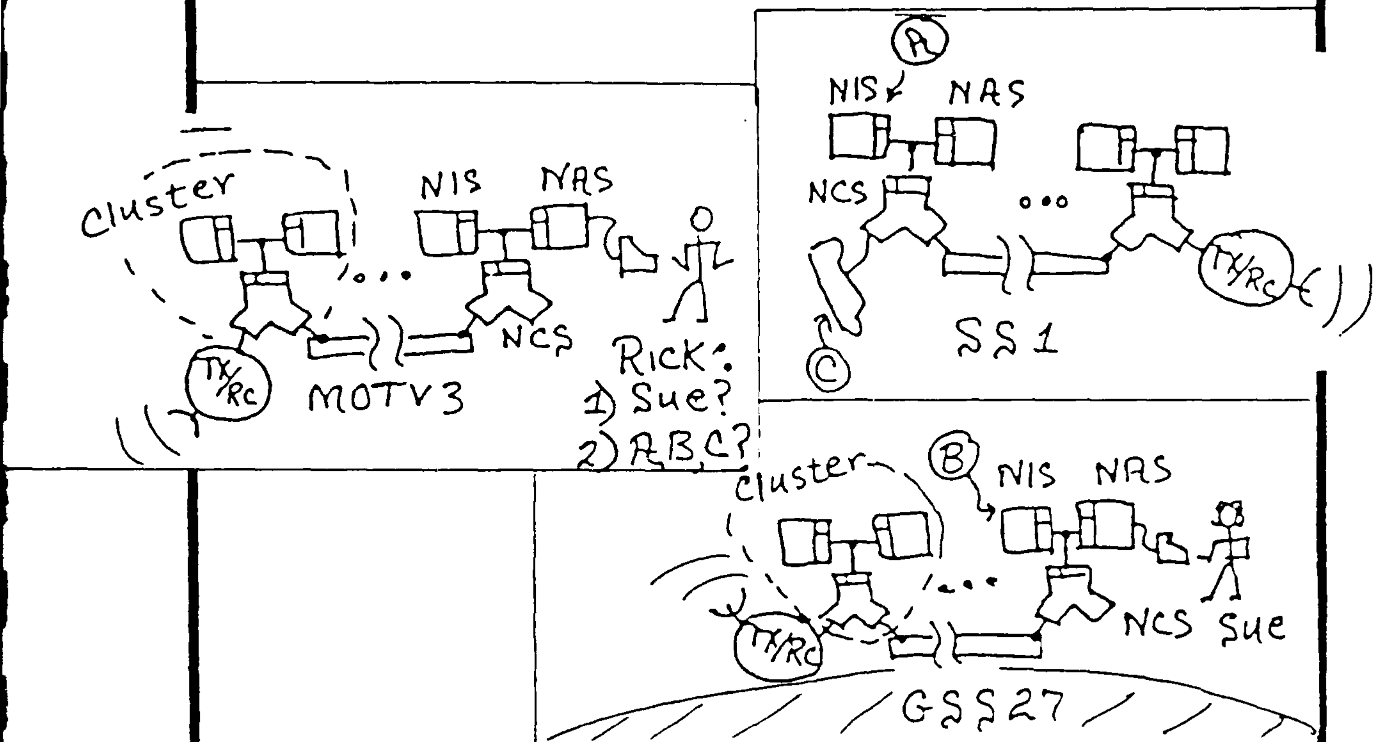
A Conceptual Model of the
Software Engineering Life Cycle





An Implementation Model
Of an SSE Based Upon the
Above Conceptual Model

PROPOSED SCENARIOS FOR SPACE STATION SYSTEMS



- 12 TYPES LAN'S
- SEVERAL LAN INSTANTIATIONS
- 23 CLUSTER: SS
- 10 YEAR DEVELOPMENT SS, 15 YEAR LIFE CYCLE, ∞ LIFE CYCLE
- TRANSPARENCY (at user friendly interface level) OF:
  - ✓LOCATION ✓PARALLELISM
  - ✓REPLICATION
  - ✓ FAULT TOLERANCE

Teaching COBOL Programmers to be Ada Software Engineers

Major Charles B. Engle, Jr.

and

Major Colen K. Willis

United States Military Academy

DoD Concern

COBOL $\longrightarrow$ Ada

Teaching

COBOL Programmers
to be
Ada Software Engineers

The Setting

- USMA Ada Summer Workshop (85-86)
- 22 DoD Programmers
- Varying Levels of Experience
- DEC Ada
- Barnes Textbook
- Bottom-up Approach
- Six Programming Assignments

Teaching
COBOL Programmers
to be
Ada Software Engineers

Mindset

- COBOL = THE Language
- Approach
  - Dependence on O/S
  - Little regard for Software Lifecycle
- Cannot Prevent but Must Handle

Fundamental Computer Science Concepts

- Generally Lacked Knowledge of Basic CS Concepts
- Programmed through Memorization
- Most Used Small "Comfortable" Subset of COBOL
- Had to Address on Day 3

Scope and Visibility

- First "un - COBOL- like" Concept => Big Problem!
- Stuck on "Flat" Structure
- Translated Concept Through COBOL Record Levels

Data Types

- Ada Typing Philosophy => No Problem!
- Strong Typing / Name Equivalence => Little Problem!
- Predefined types => Little Problem
  - COBOL Types / PIC Clause
- User-defined Types => Bigger Problem!
  - Enumerated Types, Arrays, Records

Teaching
COBOL Programmers
to be
Ada Software Engineers

Building Programs

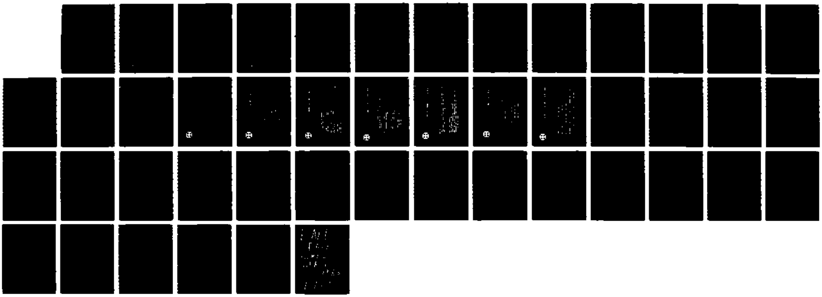
- "Patchers" vs "Builders"
- Lacked Knowledge of SE Concepts
- Inserted SE Concepts Informally
- Emphasized Modularity, Separate Compilation, Reuseability

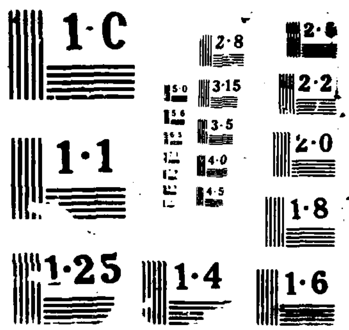
Abstraction

- Data Abstraction => New Concept / Little Problem!
- Procedural Abstraction => Bigger Problem!
  - Translated Through COBOL Paragraphs
  - Struggled with Functions, Parameters

AD-A189 650 ANNUAL ASEET (ADA SOFTWARE ENGINEERING EDUCATION AND TRAINING) SYMPOSIUM (SLIDES) HELD IN DALLAS TX ON 9-11 JUNE 1987(U) ADA JOINT PROGRAM OFFICE ARLINGTON VA 4/4

UNCLASSIFIED 11 JUN 87 F/G 12/5 NL




1·0
2·8
2·5
3·15
2·2
3·5
2·0
4·0
1·1
4·5
1·8
1·25
1·4
1·6

Dynamic Data and Recursion

- Concept Difficult to Grasp (as Expected)
- Should Have Avoided in Two Week Course

Teaching
COBOL Programmers to be Ada Software Engineers

Ada's Advanced Features

- Packages / Generics => Little Problem!
  - Built on Instantiation of I/O Packages
- Exceptions => Little Problem!
  - COBOL "On End" Analogy
- Tasking => Big Problem!
  - Concept Difficult to Grasp
  - Ran Out of Time

Teaching
COBOL Programmers
to be
Ada Software Engineers

Advance Assignment

- Required Each Student to Complete Final Assignment in COBOL Before Course
- Compared Ada vs COBOL Implementations as Part of Final Assignment

Teaching COBOL Programmers to be Ada Software Engineers

Summary

- Unscientific Analysis
- Major Conclusions
  - Translate Ada Concepts through "COBOL-eze"
  - Offer Pre-course in CS Fundamentals
  - Base Scope of Course on Amount of Time Available (i.e. Time for Tasking?)
  - Focus on Ada Effort Not Just Language

TEACHING SOFTWARE ENGINEERING

IN A FIRST ADA COURSE

by Robert C. Mers

N. C. Agricultural and Technical State University

SOFTWARE ENGINEERING PRINCIPLES

Data Abstraction and Information Hiding

Encapsulation of Related Data

Strong Typing

Separate Compilation and Use of Libraries

Readability and Style

Data Integrity

Exception Handling

Generics

Tasking

COURSE OVERVIEW

I. INTRODUCTION

History of Ada to the current moment.
Rationale for Ada
Software Engineering Principles
Overview of Packages, Subprograms, and Generics
    (to enable students to use and understand TEXT_IO)

II. SCALAR TYPES, CONTROL STRUCTURES, SUBPROGRAMS

Predefined Scalar Types and Operations
Discrete Types, Attributes, and Operations including
    Enumerated Types, SubTypes, Derived Types
Type Conversion

Decision Structures
- IF-THEN-ELSE
- CASE

Loop Structures
- FOR Loops
- WHILE Loops
- Basic Loops and EXIT Statements

Subprogram Features
- Default Parameters
- Named and Positional Notation
- Recursion
- Overloading of Names, Operators

III. COMPOUND DATA TYPES

DECLARE Blocks -- for Run Time Array Processing

Unconstrained and Constrained Arrays
Array Attributes and Operations
Slices, Aggregates, Strings
Array Type Conversion

Simple Records, including Aggregates and Default Values

IV. PACKAGES AND SEPARATE COMPILATION

Library and Secondary Units, including SubUnits
Compilation Orders

Private Types, Normal and Limited
Deferred Constants

Scope and Visibility of Names, including
Dot Notation and Renaming

V. ADVANCED ADA -- GENERICS AND EXCEPTIONS

Predefined and IO Exceptions
Declaration, Raising, and Handling of Exceptions
Propagation

Generic Subprograms and Packages
Generic Formal Parameters including
Objects, Types, and Subprograms
Generic Instantiation
Applications such as Sort, Stack Package

VI. ADVANCED TYPES

Discriminated Records including applications
to Variable Sized Arrays and Variant Records
Constrained and Unconstrained Records

Access Types

User Defined Real Types, Floating and Fixed
Representation in Memory
Operations and Attributes

VII. TASKING

Task Semantics, Flow of Control
Task Specifications and Bodies
Task Rendevous
Entry Declarations, Entry Calls, Accept Statements
Selective Wait statements

OBJECTIVES OF INDIVIDUAL PROGRAM ASSIGNMENTS

Hands-On Experience with the Following Constructs:

FIRST THIRD OF COURSE

Predefined and User Defined Scalar Types
Control Structures
TEXT_IO Features including Enumeration_IO
Use of Existing Packages
Separate Compilation

MIDDLE THIRD OF COURSE

Compound Data Types including
    Unconstrained Arrays and Simple Records
DECLARE Blocks
Array and Record Aggregates
Array Attributes
Package Design and Implementation
Overloaded Subprograms and Operators

IF TIME PERMITS
Access Types
Discriminated Records
Exception Handling
Private types

OBJECTIVES OF TEAM PROJECTS

Hands-On Experience Integrating the Major
Software Engineering Aspects of Ada

REQUIREMENTS: For All Projects

200-400 Lines of Code
Extensive Modularization
A Small System, Solution of Significant Problem,
    or Complete Implementation of Data Structure

Extensive External Package Design
Data Abstraction and Information Hiding
Exception Handling

REQUIREMENTS -- Highly Recommended

Data Integrity - Use of Private Types
Reusability - Use of Generics

EXAMPLES OF TEAM PROJECTS

(1). GENERIC TRANSCENDENTAL FUNCTION PACKAGE
For Floating Point Types - Included SQRT, EXP, LOG

(2). (GENERIC) QUEUE PACKAGE
Included Enqueue, Serve, Delete from within Queue.
Used Access Types

(3). GENERIC NUMERICAL ANALYSIS PACKAGE
Trapezoidal Rule applied to Transcendental Functions

(4). BINARY SEARCH TREE PACKAGE
Recursive Traverse, Iterative Insert and Delete

(5). CIRCULARLY DOUBLY LINKED LIST PACKAGE
Insert, Delete, Locate, Traverse. Access Types.

(6). MATRIX OPERATIONS PACKAGE
Menu Driven. Addition, Multiplication, Determinant,
Transpose, and Inverse

(7). NESTED DISCRIMINATED RECORDS
Employee Records, Two Discriminant Fields

RESOURCES - STRENGTHS AND WEAKNESSES

COMPILER -- NYU Ada Ed Version 1.7

Faster than previous NYU versions.
Accurate diagonostic in list file.
Separate Compilation, Binding, Execution

No Special Utilities such as Debugger, Math
   Libraries, Debugger, special Editor
Full of Errors
   Does not sort strings.
   Does not recognize End_of_File when
      File created from Editor
   Run Time Aborts for Generic Subprogram
      Formal Parameters, Generic Access Types
   Undocumentated Exceptions such as
      SYSTEM ERROR

   DEC VAX ADA Available in Fall 87

TEXTS AND REFERENCES

PRIMARY TEXT: S. Young, <u>An Introduction to Ada</u>
Excellent in Completeness, Readablilty, Applications, Consistent Clarity, True to Spirit of Software Engineering

OTHER CANDIDATES:

(1) <u>Software Engineering Using Ada</u> , Booch.
Ada features buried too deeply, somewhat advanced for first course.

(2) <u>Understanding Ada, a Software Engineering Approach</u> , Bray.
Readable, Bottom Up approach, too sketchy & rapid treatment of types

Use of the LRM Expected.

RECOMMENDATIONS

(1). Do not slight Syntax. Primary or secondary text should give complete elementary treatment of Ada.

(2). Emphasize packages, separate compilation, data abstraction, and robustness early.

(3). Spiral approach on packages, generics, exception handling, private types. Early exposure, later more in depth treatment.

(4). Application of private types in data structures such as stacks and queues, exercises in corrupting these structures without private types.

(5). Team Projects given 2/3 rds through course; in depth treatment of software engineering projects concurrent with student implementation.

Parks College

SAINT LOUIS UNIVERSITY

ADA® EDUCATION AND THE NON COMPUTER SCIENTIST

DR. CHARLES C. KIRKPATRICK

DR. PAUL B. KNESE

Parks College
SAINT LOUIS UNIVERSITY

OVERVIEW

- BACKGROUND
- START-UP LESSONS
- MAINTENANCE LESSONS
- CONCLUSIONS
- MESSAGE

Parks College
SAINT LOUIS UNIVERSITY

BACKGROUND

- PARKS COLLEGE - SLU SCHOOL OF AEROSPACE TECHNOLOGY
  - 9 BS DEGREE PROGRAMS
  - NO COMPUTER SCIENCE
- ALL DEGREE PROGRAMS REQUIRE COMPUTER
  - FORTRAN, Pascal, BASIC
- AVIONICS GENERATED INTEREST IN ADA
  - NATURE OF THE BUSINESS
  - MOST GRADUATES GO TO DoD CONTRACTORS / MILITARY
- COLLEGE COMPUTER INADEQUATE
  - HP 3000
  - NO ADA COMPILERS AVAILABLE

Parks College
SAINT LOUIS UNIVERSITY

START-UP LESSONS

- ADA NEEDS A STRONG ADVOCATE
  - ADMINISTRATORS DON'T UNDERSTAND
  - EXPENSIVE
- SELECTION OF COMPILER IS DIFFICULT
  - HARDWARE-DEPENDENT
  - FEW VALIDATED COMPILERS
- TEXT SELECTION IS DIFFICULT
  - ADA TEXTS WRITTEN BY COMPUTER SCIENTISTS
  - NEED AN "ADAM OSBORNE"
- SOME DEALS ARE TOO GOOD TO BE TRUE
  - TRY BEFORE YOU BUY
  - GET REFERENCES

Parks College
SAINT LOUIS UNIVERSITY

MAINTENANCE LESSONS

- COMPILER
  - NON-VALIDATED COMPILERS GIVE STUDENTS TOO MANY EXCUSES
  - LONG COMPILE TIMES DISCOURAGE EXPERIMENTATION
  - MAKE NO CHANGES DURING COURSE

- STUDENTS AND LEARNING
  - INEXPERIENCED STUDENTS CATCH ON QUICKLY
  - PASCAL NOT A PREREQUISITE LANGUAGE
  - BASIC AND FORTRAN PROGRAMMERS SOMETIMES MORE RESPONSIVE
  - LEARNING ADA IS MORE THAN LEARNING NEW SYNTAX
  - QUIZZES ARE NECESSARY

Parks College
SAINT LOUIS UNIVERSITY

CONCLUSIONS

- NEED A FAST, VALIDATED COMPILER
  - TURBO PASCAL IS A GOOD GOAL
- ADA IS FOR EVERYONE
  - COURSE WAS WELL-RECEIVED INITIALLY
  - COURSE NOW IN DEMAND

Parks College
SAINT LOUIS UNIVERSITY

ADA IS FOR EVERYONE!

- COMPLEX -- BUT ALL LANGUAGES ARE COMPLEX
- ADA SUITABLE AS A "FIRST" LANGUAGE
- USE ADA TO LEARN SOFTWARE DEVELOPMENT
- ENGINEERS USING ADA IN SCHOOL WILL DEMAND ADA IN INDUSTRY
- APPLICATIONS SOFTWARE WILL BE WRITTEN BY ENGINEERS

Ada* in the Undergraduate Curriculum
at
Saint Mary College

Victor A Meyer

*Ada is a registered trademark of the U.S. Government (Ada Joint Program Office)

SAINT MARY COLLEGE

1. Located in Leavenworth, Kansas
2. Small private liberal arts college
3. Only women's residential college in the region
4. First computer course offered in 1977
5. Computer Science major first offered in 1981
6. Ada first taught in Fall, 1985

COMPUTER SCIENCE GOALS
AT
SAINT MARY COLLEGE

1. Develop quality programmer/analysts

2. Utilize state-of-the-art hardware & software

   Example: The Ada language

3. Provide for computer needs of local community

   Example: Fort Leavenworth, KS

WHY TEACH ADA
AT
SAINT MARY COLLEGE

1. It is the most recently developed language
2. There is a demand for Ada programmers in the area
3. Colleges and universities establish computer trends
4. Students can practice the latest programming concepts
5. Small class sizes - Computers not overloaded
6. Ada graduates can more easily adapt to other languages

ADA CURRICULUM
AT
SAINT MARY COLLEGE

Required Courses:

General Programming I & II - Ada fundamentals

File Constructs - file handling techniques

Data Structures - data abstraction, generics

Other Ada Related Courses:

Software Engineering

Special Topics - parallel processing

LESSONS LEARNED

1. Students accept 3 minute compilation times

2. More demand for Ada graduates than Pascal graduates

3. Students comprehend Ada the same as Pascal

4. Pascal students readily pick up Ada syntax

5. Pascal students write Ada programs with a Pascal dialect to them

LESSONS LEARNED (Continued)

6. Students prefer Ada over Pascal

7. Algorithms are easier to understand in Ada

8. Obtaining grants to support Ada undergraduate curriculums is difficult

9. Ada does not force students to use the language properly (e.g. new integers)

10. Few textbooks teach Ada as a first language

CONCLUSION

In our opinion, the only good reason for not teaching Ada at the undergraduate level is the problem of the slow and inefficient compilers that are currently on the market.

GOVERNING CONSIDERATIONS

LIMITED MEANS

-- INITIALLY ONLY SUBSET COMPILER AVAILABLE

-- DELIVERY DATE OF VALIDATED COMPILER UNCERTAIN

-- RESTRICTIONS ON AVAILABLE HARDWARE

OTHER PRIORITIES

LIMITED OBJECTIVES

BARRETT

GOALS

1. TEST SUITABILITY OF ADA FOR USE IN A PROJECTS COURSE

2. EXPLORE THE CONCEPT OF AN ADA SHORT COURSE AND FACILITATE THE INCLUSION OF ADA IN SURVEY OF LANGUAGES COURSE

3. PROVIDE EXPERIENCE IN INSTRUCTION IN ADA

4. DEVELOP CADRE OF ADA PROGRAMMERS TO PARTICIPATE IN FUTURE PROJECTS

ORGANIZATION

1. ORGANIZED AS SPECIAL TOPICS COURSE

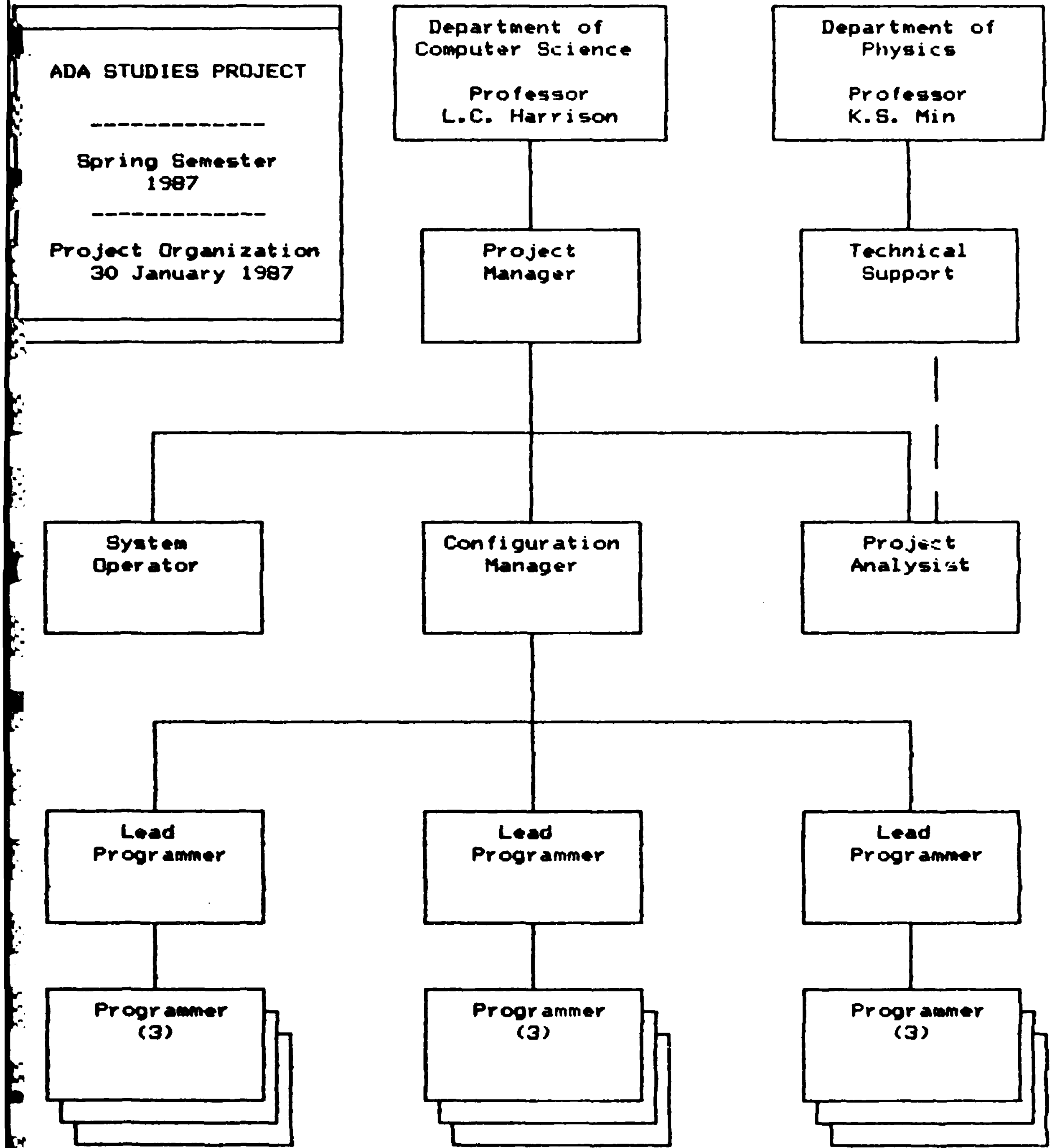
   -- FLEXIBILITY IN COURSE CONTENT AND ORGANITION

2. PROJECT ORIENTED

3. STUDENT ORGANIZATION AND MANAGEMENT

4. GRADUATE AND UNDERGRADUATE SPECIAL TOPICS SECTIONS

   – GRADUATE STUDENTS PROVIDED LEADERSHIP FOR UNDERGRADUATES


ADA STUDIES PROJECT
Spring Semester 1987
Project Organization 30 January 1987
Department of Computer Science
Professor L.C. Harrison
Department of Physics
Professor K.S. Min
Project Manager
Technical Support
System Operator
Configuration Manager
Project Analysisist
Lead Programmer
Lead Programmer
Lead Programmer
Programmer (3)
Programmer (3)
Programmer (3)

CONCLUSIONS

LIMITATIONS OF VALIDITY OF CONCLUSIONS

-- RIGOROUS STUDY DESIGN NOT ATTEMPTED

-- "ONE-SHOT" CASE STUDY

-- OBSERVERS ALSO PARTICIPANTS

-- CONCLUSIONS CONSIDERED PRELIMINARY

PRINCIPAL CONCLUSIONS

-- ELEMENTARY FEATURES OF ADA ARE EASILY LEARNED, SUGGESTING SUITABILITY FOR USE IN INTRODUCTORY COURSES

-- ADVANCED FEATURES MAY BE LESS EASILY LEARNED, SUGGESTING THE NEED FOR AN ADDITIONAL COURSE IN ADVANCED ADA, OR A COURSE IN SOFTWARE DEVELOPMENT CONCEPTS AS A PREREQUISITE TO A PROJECTS COURSE

-- THE INTRODUCTION OF STANDARDIZED, REUSABLE, PROJECT ORIENTED TOOLSETS AND INSTRUCTIONAL MODULES WILL ENHANCE THE EFFECTIVENESS OF PROJECTS COURSES

Teaching Ada* in the University

Presented at The SECOND ANNUAL ASEET
SYMPOSIUM
Dallas, Texas
June 11, 1987

by

Cliff Layton, Codirector
The Rogers Institute of Software Engineering
The RISE
Will Rogers and College Hill
Claremore, Ok 74017
918-341-7510–286

* Ada is a registered trademark of the U.S. Government.

OVERVIEW

I. Speaker's Context

II. Ada Language Characteristics Supporting use in Higher Education

III. ASEET Implementation in Higher Education

IV. Orgs. Which Aid ASEET Higher Ed. Implementation

V. Overcoming Resistance to ASEET Implementation in Higher Ed.

VI. Conclusion

I. Speaker's Context

A. ASEET

1. College and University: Mostly Soph.

2. Business/Industry

a. 1 Week Courses

b. OJT

B. "Real Ada and Software Engineering Work" in Business and Industry

1. Ada Systems Evaluation

2. Ada Software Engr. Proposal Writing

3. Ada Software Engr. Contract Working

C. Books Used in Courses

1. Barnes, J.G.P.: <u>Programming in Ada</u>, Addison-Wesley, Reading, MA, 1982.

2. Booch, Grady: <u>Software Engineering with Ada</u>, Benjamin/Cummings, Menlo Park, CA, 1983 and 1987.

3. Cohen, Norman H.: <u>Ada as a Second Language</u>, McGraw-Hill, N.Y., 1986.

4. EVB Software Engineering Inc.: <u>An Object Oriented Design Handbook for Ada Software</u>, Fredrick, MD, 1985.

5. Saib, Sabina: <u>Ada: An Introduction</u>, Holt, Rinehart and Winston, N.Y., 1985.

D. Ada Compilers Used in Courses

1.	Janus	IBM-PC, 1983
2.	Meridian	IBM-PC, 1987
3.	Telesoft	VAX, 1983-85
4.	VAX (DEC)	VAX, 1986-87

II. Ada Language Characteristics Supporting use in Higher Education

A. Strong-Typing and Programmer-Defined Typing

B. Block-Structured

C. Procedural

D. Module-Oriented

E. Expression, Control, Functional, Procedural, Type and Process Abstraction

F. Object-Oriented

1. Encapsulation

2. Well-Defined Interfaces

3. Loose Coupling

4. Inheritance

G. Analysis and Design Level Interfacing

H. Computer and Hwd. Level Interfacing

I. Support for Phases of Software Development

J. Support for Software Engineering

III. Ada Implementation in Higher Education

A. Ada and Software Engineering Education Should be Done in Concert

1. The Goals and Principles of Software Engineering were the Design Requirements of Ada

2. Ada and Software Engineering are Mutually Supportive

3. Software Development Productivity is Optimized by Ada Embedded in Software Engineering

B. Ada use in Higher Ed. is Pedagogically Sound

1. Consistent with Model CS and DP Curricula

a. Introduction to Programming Methodology

b. Program Design and Implementation

c. Algorithms

d. Data Structures

e. Software Engineering

B. Ada use in Higher Ed. is Pedagogically Sound

f. Operating Systems

g. Artificial Intelligence

h. Numerical Analysis

2. Facilitates Structure and Understandability

3. Supports all Phases of Sw. Development

C. Ada use in Higher Ed. is Consistent with "Real World" Applicability

1. Ada has High U. S. Dept. of Defense Potential

2. Ada has High Commerical Potential

D. Ada use in Higher Education is Increasingly Supported by Texts and Affordable Software

E. Ada Courses are Offered in More Than 100 Institutions of Higher Education in the U.S.

IV. Orgs. Which Aid ASEET Higher Ed. Implementation

A. ACM SIGAda 609-234-8510

B. ACM SIGAda Education Comm. 201-922-6323

C. Ada Information Clearing House 703-865-1477

D. Ada Joint Program Office 202-694-0210

E. ASEET 601-377-2030

F. Software Engineering Institute 412-268-7700

G. Higher Educational Institutions Offering Ada and Software Engineering Courses

V. Overcoming Resistance to ASEET Implementation in Higher Education

A. Consult the Organizations in IV. Above

B. Stress the Growth and Growth Potential of Ada and Software Engineering

C. Build Ada and Software Engineering Bridges Between Education, Commercial Business and Industry, U.S. DOD-Related Business and Industry, Political Orgs., and Professional Orgs.

1. Visit, Inform and Recruit

2. Get Involved in a Local Ada SIG

D. Secure Extra Ada and Software Engineering Funding

1. Grants

2. Contracts

3. Product Donations and Discounts

E. Gain the Support of Top-Level Administration

F. Gain and Use the Best Affordable Ada and Software Engineering Products

G. Stress the Pedagogical Consistency of ASEET

VI. Conclusion

A. Ada Language Characteristics Support use in a Wide Variety of CS, DP and Other Higher Education Courses at All Levels

B. Ada Should be Used in a Software Engineering Context

C. Ada and Software Engineering Potential Supports ASEET Implementations in U.S. Higher Ed.

D. Over 100 (Pedagogically Consistent) Ada Implementations are in Place in U.S. Higher Ed.

E. ASEET Educational Materials, Including Affordable Software, are Increaingly Available

F. Organizations Assisting ASEET Implementations in Higher Ed. are in Place and Very Helpful

G. Resistance to ASEET Implementataions in Higher Ed. is Formidable but Can be Overcome

END
DATE
FILMED
APRIL
1988
DTIC